パソコンプログラマのためのクリエイティブマガジン

1991年5月1日発行（毎月1回1日） 第1巻 第5号 通巻5号

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1991 5 MAY

定価980円



特集

基礎力としてのアセンブラ

コンピュータを理解し，その能力をひきだすために

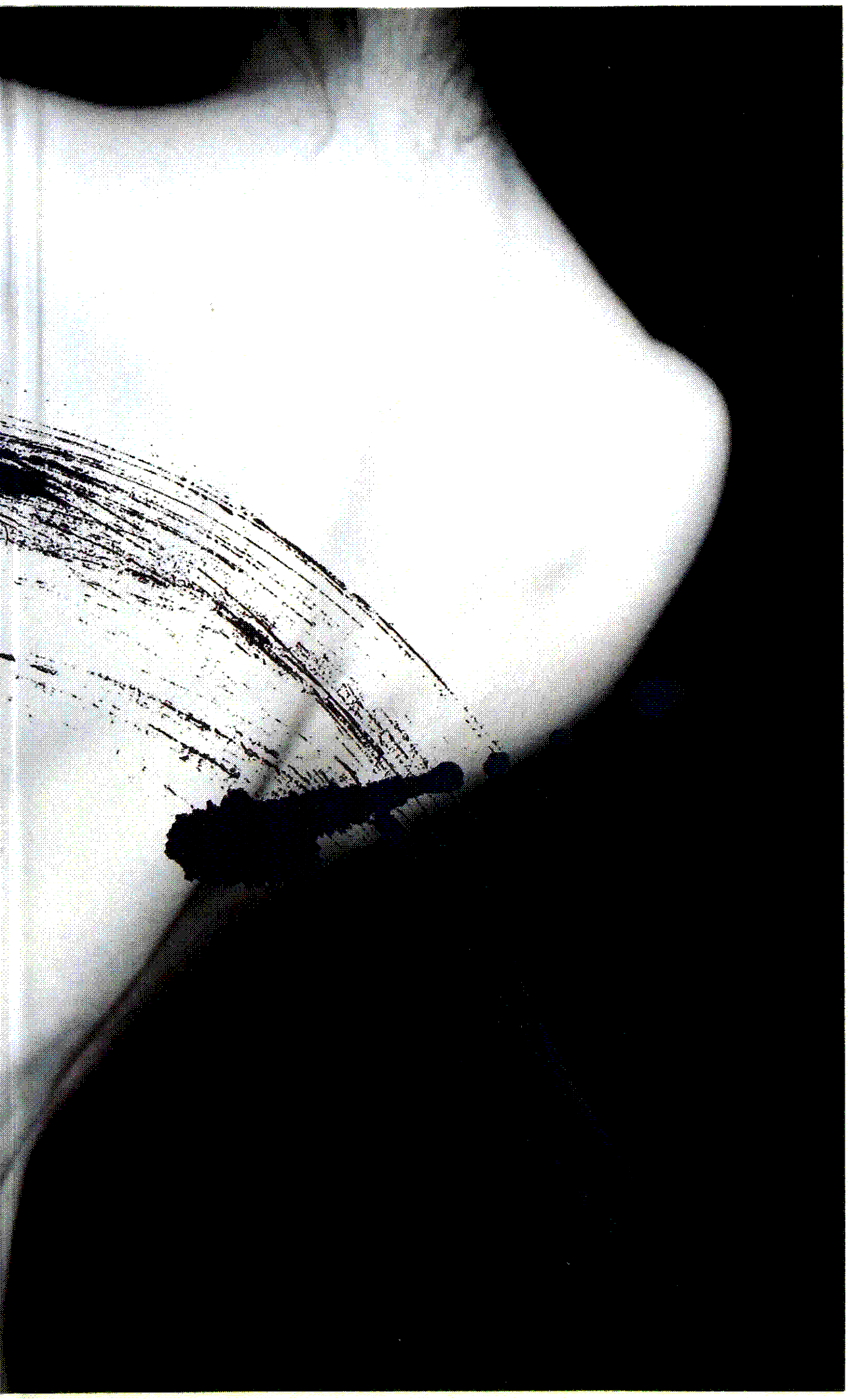
付録:5"2HDフロッピーディスク

本誌掲載プログラム／綴じ込み付録テキストファイル（8086インストラクション一覧）／クイズ・ランダムスクリーン 解読ツール／ビデオゲーム NUKEM 1.12／高圧縮書庫管理プログラム LHA公式公開版／シンボルラベルジェネレータ SYMGEN／マルチファイルダンプエディタ MDe

SE

Shoeisha Co., Ltd.







■新登場・Turbo Vision
もはや、アプリケーションを書く際にユーザインターフェース等の細かい部分で苦労する必要はありません。新しいTurbo Pascal 6.0に付属する、オブジェクト指向のユーザインターフェース・クラスライブラリTurbo Visionを使えば、プロ開発者レベルのすばらしいユーザインターフェースを持つアプリケーションを、驚くほど簡単に記述することができます。これによって、位置・サイズを自由に変更可能なオーバーラップウィンドゥやプルダウンメニュー、ダイアログボックス、ボタン等を駆使したマウス操作のイベントドリブン・アプリケーションを短時間で作成できます。さらに電卓、カレンダー、エディタ、時計、ディレクトリブラウザ、フォームといったアプリケーションをサンプルとしてソースコード形式で提供。これらを部品としてアプリケーションに組み込むことができます。事実、Turbo Pascal 6.0の新・統合開発環境もTurbo Visionを使って開発されています。つまり、操作性では定評のある統合開発環境と同レベルのアプリケーションが簡単に作成できる訳です。

■大きく改善された統合開発環境
新しい統合環境は、オーバーラップマルチウィンドゥをサポートし、マウス操作も可能です。エディタは、1MBまでのファイルを同時に複数オープンできます。また、内蔵デバッガでは条件ブレークポイント機能やレジスタウィンドウも追加されています。

■組み込みアセンブラ
従来のインラインアセンブル機能に加え組み込みアセンブラが使用できます。ソースプログラムにニーモニックを直接記述できるため、PASCALプログラム中でも簡単にアセンブラ言語を利用できます。

■ハイレゾルーションモードに対応
統合環境のハイレゾ31行モードでの動作、BGIやTurbo Visionでのハイレゾモードをサポート。また、H98シリーズの拡張アトリビュートにも対応しています。

■Turbo Pascal 6.0 Professional
Professional版には、タフなバグを始末するTurbo Debugger 2.01、プログラムのボトルネックを取り除くTurbo Profiler 1.0、MASMコンパチブルな高速アセンブラTurbo Assembler 2.0に加えて、プロテクトメモリを利用して巨大アプリケーションがコンパイルできるコマンドラインコンパイラが付属します。

■Turbo Pascal 6.0の動作環境
機種：PC-9801シリーズ、PC-286/386シリーズ（ハイレゾモード対応、LTは除く）／メモリ：640KB以上／MS-DOS：バージョン2.11以上／マウス：バスマウス

■価格：Turbo Pascal 6.0………29,800円（税別）
Turbo Pascal 6.0 Professional…59,800円（税別）



TURBO PASCAL® 6.0

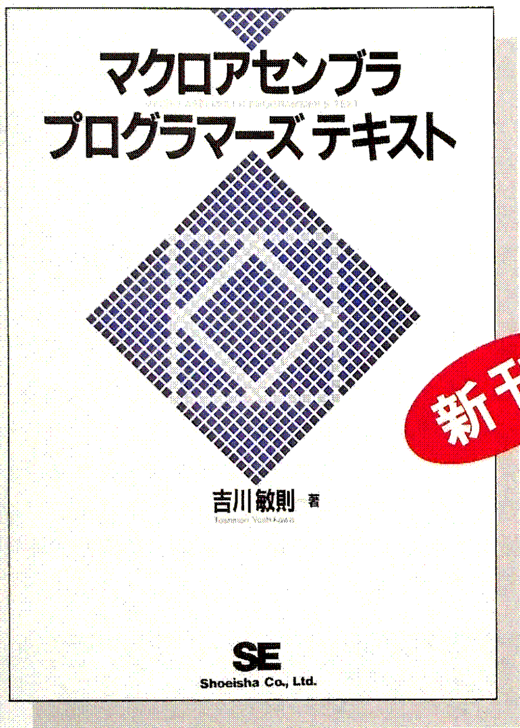
お詫びと訂正：本誌4月号掲載の広告において、Turbo Pascal 6.0 Professionalの価格表記に誤りがありました。（誤）49,800円 ──→（正）59,800円（税別）
深くお詫びするとともに、上記のように訂正いたします。

●製品に関するお問い合せは、総販売元㈱マイクロソフトウェア アソシエイツ TEL.03-3486-1411までお願いします。

株式会社ボーランドジャパン
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400㈹

〈資料請求番号001〉

SE 翔泳社が贈るプログラミングブックス



アセンブラプログラミング撤底解説宣言
アセンブラプログラマー必携の書登場!

マクロアセンブラ プログラマーズテキスト

吉川敏則 著■定価2,600円(税込)

PC-9801シリーズの代表的なCPUであるインテル社のi8086系列(i80286, i80386)および標準オペレーティングシステムのMS-DOSを活用するためのマクロアセンブラ(MASM)を使ったプログラミング書。最低限度必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説します。また後半では、実践的なアセンブラテクニックをMS-DOSファンクション・リクエストの活用法から学び、Code Viewによるデバッグ方法で締めくくります。



C言語入門書異説

Cの話

安良城勝也 著■定価1,500円(税込)

赤ん坊が言葉を学ぶときは、「はじめに文法ありき」ではない。言葉のシャワー、情景のシャワーの中でセンスが磨かれ、突如として天啓が訪れる。プログラミング言語を学ぶにも、まずセンスの醸成が必要だ。本書はC言語を学ぶ前に読む本。ソフトウェアとは何か、Cのプログラムとはどんなものか…軽いタッチで一気に読める、頭の準備体操。

SE 株式会社 翔泳社
東京都千代田区平河町2-4-14 平河町KSビル 〒102
PHONE 03-3263-0447 FAX 03-3263-4526

〈資料請求番号002〉

SE 翔泳社が贈る



QuickBASIC スタイルブック

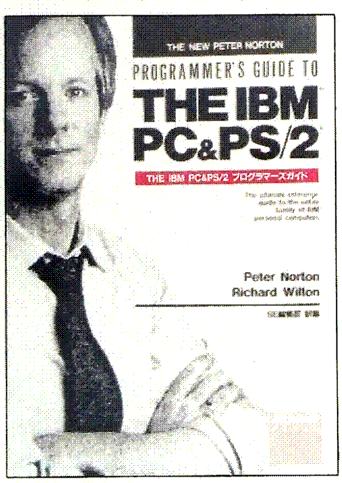
●定価2,300円

イント・ツーワン著

QBを骨までしゃぶってみませんか

数あるプログラミング言語の中でも、Quick BASICは、初心者からマニアまで幅広く使われています。この本は、少しでもわかりやすく、質の高いプログラミングを書くために一問一答式の問題集形式になっている「良いプログラミング」のための学習書です。さあ、あなたも挑戦してみてはいかがですか。

IBM PC&PS/2プログラミング3部作

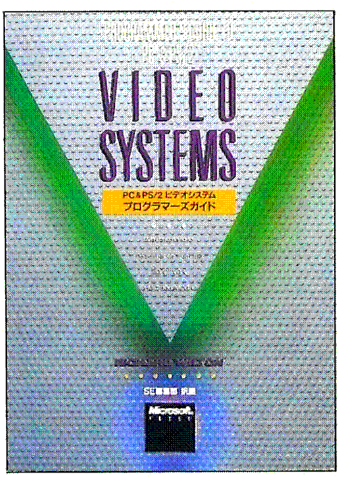


THE IBM PC&PS/2 プログラマーズガイド

●Peter Norton & Richard Wilton著

●定価5,200円

あのピーター・ノートンのベストセラー。IBMプログラマ必携のスタンダードガイダンス。



PC & PS/2 ビデオシステム プログラマーズガイド

●Richard Wilton著

●定価6,000円

IBM PCファミリーのビデオシステム完全ガイド。



プログラマーのための PCソースブック

●Thom Hogan著

●定価5,200円

IBM PC,PS/2プログラマ必携の技術資料完訳。

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

v.3.0 いよいよ登場。
プログラミングエディター新世紀。



よりパワフルで、よりエレガントな
プログラミングのために……。

●C言語で組めるマクロ 新機能
使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能
ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能
C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ
各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集
Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能
キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン（オプション）、メニュー（オプション）etc。

アメリカでも日本でも高い評価

●プログラミング向きエディターを探している人は、もう探す必要はない。（Jerry Pournelle／BYTE）
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。（日経バイト）
●既存のエディターに不満を覚えたら、これしかない。（Oh! FM）
●まるで夢のようなアンドゥだ――おもわず間違うのを期待してしまいそうだ。（The C Users Journal）

■主な機能
◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット／コピー／ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化／非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円
（消費税は含まず）

■5機種対応パッケージ
（東芝、NEC、AX協議会、IBM、富士通）
●5 2DD、3.5 2DDデュアルメディア●MS-DOS3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

〈資料請求番号004〉

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

パソコンプログラマのための
クリエイティブマガジン
月刊プログラマーズ・ページ

1991 MAY No.5

CONTENTS

特集

基礎力としてのアセンブラ

コンピュータを理解し、その能力をひきだすために



表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

Illustration: NAOKI SATO

CONTENTS

広告索引 資料請求番号

人材募集広告

(株)エルテックス
(株)ミワシステムズコンサルティング
ジィスクソフトウェア(株)
(掲載順)

ASSEMBLER

コンピュータのプログラミング言語は，たんなる数字の羅列にすぎなかったマシン語から，より人間の言葉に近い高級言語へと進化を続けてきている．高級言語を利用することで，プログラマはより複雑で高度なプログラムをより容易に生み出せるようになった．しかし，高級言語によるハードウェアの抽象化，ブラックボックス化は，プログラマに利益をもたらすと同時に“コンピュータを知らない”プログラマをも生じさせた．CPUを知らない，ハードウェアがわからない，システムレベルでのデバッグができない……．コンピュータを理解し，その能力をフルに発揮させ，自由自在に扱うためには，マシン語／アセンブリ言語の知識が不可欠である．このプログラマが直接コンピュータを操作する言語であるマシン語／アセンブリ言語をもう一度みなおしてみたい．

◆特集◆

基礎力としてのアセンブラ

コンピュータを理解し，その能力をひきだすために



1

いまどきのアセンブラプログラミング入門

アセンブラを必要とするこれだけの理由

秋山 登

アセンブラ言語を身に付けることで，プログラマはどのような利益を得ることができるのか．また，どのようなプログラミングの現場においてアセンブラプログラミングが必要とされるのだろうか．高級言語にはなしえない，アセンブラプログラミングの特徴とその利用法について解説する．

アセンブラの特徴

コンピュータの動作を完全に把握できる

高級言語(ここではC言語やPascalのようなアセンブリ言語以外の言語)でプログラミングをしていると，あるとき，ふとコンピュータはなぜこんな不思議なことができるのか，また，なぜこんな簡単なこともできないのかという疑問にぶつかることがある．筆者がアセンブリ言語をはじめたのも，このようなことがきっかけであった．それからもう20年くらいたつが，いまだにときどきアセンブラで仕事をすることがある．進歩がないといわれればそれまでの話だが，その間にアセンブラを使用することで得られた知識は，コンピュータシステム全体の理解にかなり役立っていると思っている．

さて，アセンブリ言語は，CPUを直接制御する機械語と1対1に対応する言語である．そして，アセンブリ言語で書かれたプログラムを，機械語に変換してくれるのがアセンブラと呼ばれるプログラムである．このことからもわかるように，アセンブラプログラミングには，高級言語に比べて次のような特徴がある．

①CPUの動作を完全に把握できる

機械語と1対1に対応しているため，必要であればCPUのデータシートなどから，実行時間やCPUからハードウェアに送られる信号とそのタイミングを予測することができる．

②CPUのもつ能力を最大限に活用できる

最適化はプログラマ自身にまかされているため，プログラマの能力に依存する．プログラマに十分な能力があれば，一般的に高級言語が出力する以上のコードを書くことができる．まったく同じ機能の命令コードが複数存在すれば，実行時間が短くあるいは語数が少ない命令コードに最適化してくれるアセンブラもある．また，アセンブリ言語は高級言語が普通サポートしない命令を記述したり，特殊な型のデータを処理することもできる．

③新しいCPUに対する処理系が短期間で利用可能となる

アセンブラは他の高級言語と比べるとかなり単純な処理系である．命令語の構造やオペランドの複雑さにもよるが，手慣れた人間が最小限必要な機能をもつ新たなアセンブラの処理系を作成するには1か月もあれば十分である．高度なマクロ機能をもつマクロアセンブラが手元にあれば，マクロを使用してクロスアセンブラの処理系を作成することもできる．

④多方面にわたるきめ細かい知識を必要とする

アセンブリ言語でプログラミングを行うためには，高級言語ではあまり必要のないCPUの細部におよぶ知識やOSに関する知識が必要になってくる．とくに，ハードウェアに近い部分のプログラムを作成する人達には，システムハードウェア全体の構成に関する知識，インターフェースLSIの知識，インターフェースLSIに接続される機器の知識など，ソフトウェアの知識のほかにハードウェアの知識が必要になってくる．

⑤CPUの機種に依存した言語である

CPUが一度に処理できるデータ量の上限は4ビット，8ビット，16ビット，32ビットなど機種によりさまざまである．また，ひとつひとつの命令もアセンブリ言語では書式が異なるうえ，CPUそれぞれの固有の命令も少なくない．たとえば，X＝X＋Yという演算を32ビットで行う場合，8086，68000およびZ80ではリスト1のa～cのように書く．

リスト1のa～cを一見してわかるように，記述内容もプログラムの長さもずいぶん違っている．このように，プログラムの記述内容には互換性のあるCPU以外に対してまったく移植性がない．プログラムを移植しようとすれば，全面的な書き換えが必要となる．そして，書き換えには当然移植先のCPUに関する知識が必要になることはいうまでもない．ただし，ひとつのCPUに関して十分に知識のある人であれば，他のCPUを理解することはさほど困難なことではない．また，よほど特殊なケースでないかぎりアルゴリズムの大幅な変更を迫られることは少ない．

⑥構造化プログラミングを適用できない

高度なマクロを駆使して，高級言語風の構造化プログラミングを行うことは不可能ではないが，マクロの機能およびコードの効率上の問題もあり，あまり一般的ではない．アセンブリ言語による通常のプログラミングでは，高級言語で見られるIF～THEN～ELSE文やWHILE文といった制御文と同等の機能を実現しようとすると，必然的にジャンプ系の命令を使用せざるを得ない．

たとえば，リスト2aに示すC言語の

アセンブラ用語集
ANK　Alphabet（英字）・Number（数字）・Kana（カタカナ）を意味し，8ビットで表現できる文字セットのこと．

リスト1　32ビットの演算

```
a.  X=X+Y(8086の場合)

        MOV     AX,WORD PTR Y
        ADD     WORD PTR X,AX
        MOV     AX,WORD PTR Y+2
        ADC     WORD PTR X+2,AX

X       DD      ?
Y       DD      ?

b.  X=X+Y(68000の場合)

        MOVE.L  Y,D0
        ADD.L   D0,X

X       DS.L    1
Y       DS.L    1

c.  X=X+Y(Z80の場合)

        OR      A
        LD      HL,(X)
        LD      DE,(Y)
        ADC     HL,DE
        LD      (X),HL
        LD      HL,(X+2)
        LD      DE,(Y+2)
        ADC     HL,DE
        LD      (X+2),HL

X       DS      4
Y       DS      4
```

リスト2　C言語とアセンブリ言語の比較

```
a. Cのwhile文

        while(sts) {
        /* while文内の処理*/
        }

b. アセンブリ言語によるwhile文

WH0002: MOV     AX,STS[BP]
        AND     AX,AX
        JZ      WH0001
;       /* while文内の処理*/
        JMP     SHORT WH0002
WH0001:
```

while文を8086のアセンブリ言語で書くと，リスト2bのようになる．リスト2bから明らかなようにアセンブリ言語では，JZとJMP命令を使わないとwhile文と同等な処理ができない．ただし，アセンブリ言語のステップごとに眺めれば確かにその通りなのだが，構造化を意識してブロックごとにまとまったコーディングをする癖をつければ，JMP系の命令による混乱は避けられるものである．

⑦プログラムの記述量が多くなる

高級言語では1行で書けても，アセンブリ言語で書くと数行から数十行になることは珍しくない．そのため同じ処理を記述するにしても，アセンブリ言語で書くとかなりの記述量を必要とすることになる．記述量が増えるとそれだけミスが入りやすくなり，プログラムを読むことも容易ではなくなる．これは上記⑤の例を見れば明らかであろう．

⑧可読性がよくない

マクロ命令の使用，ラベルの工夫，コメントなどによりある程度改善はできるが，読みやすさという点では直感的な記述ができないため高級言語にははるかに及ばない．⑦で述べたが，プログラムが長くなることも読みやすさを阻害する一因である．

以上のような特徴をもつアセンブリ言語は，具体的にどのようなものなのか，そしてそれを使いこなすにはどのような知識を求めたらよいのかを以下に紹介する．

アセンブラはどこに生きているか？

◆

高級言語にはなしえないことを実現する

マイクロプロセッサが世に出てからしばらくの間，マイクロプロセッサに対するプログラミングは圧倒的にアセンブリ言語やBASICを使用することが多かった．このような初期のころは，高級言語による処理系が少なく，ハードウェアも高級言語の出力するコードを生かしきれるほど充実していなかったからである．ところが，最近では32ビットプロセッサがパソコンでは常識になりつつあり，ハードウェアは初期のころの10倍から数10倍の速度で動作し，使用できるメモリの量もMBの単位が普通になっている．処理系のほうもC言語をはじめとして数多くのものが普及し，高級言語の出力するコードに対する最適化技術はずいぶん高度なものになってきている．

このような状況では，アセンブラの出番はないように感じられるかもしれないが，そうではない．たしかに，パソコン用のアプリケーションプログラムの開発は，ほとんど高級言語で行われているのが現状である．しかし，高級

アセンブラ用語集
BCD　Binary Coded Decimal（2進化10進）の略．桁数が大きくなると本来の10進数とは多少の誤差が生じる．

言語では実現できなかったり，実現できるにしても，性能上の問題があるプログラミングの分野は存在するのである．次にその例をいくつかあげてみよう．

速度に依存する場所

ハードウェア設計の都合上，プログラムでタイミングをとったり，あるいはコンピュータ外部の状態の時間的な変化に対応してプログラムを動作させるような場合，プログラムのある区間の実行時間を一定時間以内に保つ必要がある．この一定時間が，処理の内容とCPUの速度を比べて余裕のない場合，アセンブリ言語を使用しなければならないことがある．

身近な例として，D/Aコンバータへ音声を出力するためのデータを送ることを考えてみよう．5KHzの帯域幅を確保するためには，10KHzのサンプリング速度，すなわち100マイクロ秒に1回データをD/Aコンバータに送ってやる必要がある．D/Aコンバータに出力するための出力レジスタは16ビット長で，0D0H番地（下位バイト）と0D1H番地（上位バイト）にあるものとする．またD/Aコンバータのタイミングをとるため，0D2番地のビット7は100マイクロ秒ごとに1となり，0D2番地の内容を読み込むと0になるものとする．

これを8086のアセンブリ言語で書くとリスト3のようになる．DS:SIには出力するデータへのポインタが，CX:BXにはデータの残りワード数が入っているものとしている．各行のコメントは命令を実行するのに必要なクロック数である．

リスト3から計算すると，0D2H番地のビット7がオンになってから最後のRET命令の1つ前のJNZ命令によりTONE01番地に戻る間の実行時間だけで最良の場合94クロック，最悪の場合で118クロックである．プリフェッチがすべてうまくいくとは限らないので，この間にある18ワードをフェッチする時間72クロックを念のため加算すると最悪でも190クロックとなる．

8086が8MHzのクロックで動作していると，1クロック当たり125ナノ秒であるから，この場合，最悪でも実行に必要な時間は23.75マイクロ秒であり，満足な動作を期待できることがわかる．ただし，この数値はメモリアクセスも入出力アクセスも，待ち時間がないものとしているので，そのような待ち時間を必要とするシステムでは，さらに時間がかかる可能性がある．

リスト3　実行時間の計算

```
0000  E4D2     TONE01: IN    AL,0D2H ; 10
0002  2480             AND   AL,80H  ;  4
0004  74FA             JZ    TONE01  ;  4/16
               ;
0006  8B04             MOV   AX,[SI] ; 13
0008  E6D0             OUT   0D0H,AL ;  8
000A  8AC4             MOV   AL,AH   ;  2
000C  E6D1             OUT   0D1H,AL ;  8
000E  83C602           ADD   SI,2    ;  4
0011  7307             JNC   TONE02  ;  4/16
0013  8CD8             MOV   AX,DS   ;  2
0015  80C410           ADD   AH,10H  ;  4
0018  8ED8             MOV   DS,AX   ;  2
001A  83EB01   TONE02: SUB   BX,1    ;  4
001D  83D900           SBB   CX,0    ;  4
0020  8BC3             MOV   AX,BX   ;  2
0022  0BC1             OR    AX,CX   ;  3
0024  75DA             JNZ   TONE01  ;  4/16
0026  C3               RET           ;  8
```

記憶容量が制限される場合

電化製品や小規模の制御装置に組み込まれる4ビットや8ビットのマイクロプロセッサには，メモリ容量の制限が厳しいものが多い．これは，小型化と低コストを追求するため仕方のないことである．このようなマイクロプロセッサのために，ある程度のプログラムを書こうとすると，アセンブリ言語を使用する以外に手がないのが普通である．

たとえば，インテル社の8051の場合，プログラムメモリ（マスクROM）4KB，データメモリ（RAM）128バイトで，スタックはデータメモリの128バイトの範囲内に限られる．このようなマイクロプロセッサは，身の回りの機器の中に意外と多く使用されている．

通常の実行形式以外のプログラムファイルが必要な場合

高級言語の処理系は，最終的にそれがターゲットとしているOSのもとで走行する実行形式のファイルを出力する．これはあたりまえのことであるが，OSの中には通常の実行形式以外に特別なプログラムファイルを受けつけるものがある．この代表的な例が，MS-DOSのデバイスドライバである．MS-DOSのデバイスドライバは，MS-DOSの実行形式ファイルである.EXEファイルとも.COMファイルとも形式が異なっている．

たとえば，キャラクタ型のデバイスドライバには，次に示す18バイト固定のデバイスヘッダと呼ばれるデータが必要である．そして，実際のプログラ

アセンブラ用語集
NEAR型　分岐命令やサブルーチンコールでジャンプ先が同一セグメント（64 KB）内にある場合を指す．セグメント外にジャンプ先がある場合はFAR型という．

ムはデバイスヘッダの直後からはじまる．そのようすをリスト4に示す．

高級言語でリスト4のようにデバイスヘッダを配置することは無理で，少なくともこの部分はアセンブリ言語で記述する必要がある．

もうひとつの問題は，ストラテジルーチンの呼び出し規約で，ES:BXにデバイスドライバに対するコマンドパラメータのアドレスが引き渡されることである．高級言語で作成するプログラムでは，普通レジスタを自由に制御することができないのでデバイスドライバに限らず，レジスタでパラメータを引き渡されたり，引き渡したりする場合にもこの問題が発生する．

高級言語のサポートしない型のデータ処理を行う場合

BCD形式の数値をサポートしている言語としてCOBOLやPL/Iがあるが，FORTRANやCではBCD形式の数値をサポートしていない．このように，BCD形式の数値をサポートしていない言語でBCD形式の数値を演算するサブルーチンや関数を作成するのは不可能ではないが，かなりめんどうなものである．加減算の範囲であれば，アセンブリ言語で作成するのが簡単であるし，速度やサイズの面でも有利である．

MS-C 6.0のインラインアセンブラで書いた例をリスト5にあげておこう．この例のbcdaddという関数は，2つのnバイトのBCD形式の数値へのポインタxとyを受け取り，2つの数値を加算して，結果をnバイトのBCD形式の数値でxの指す領域へ格納する．BCD形式の数値と同様，サポートされていない精度の純2進数形式の整数やサポートされているものと形式の異なる浮動小数点数を効率や互換上の理由で処理したい場合，アセンブリ言語は有効である．

高級言語が使えない場合

前記の記憶容量の制限が厳しい場合にも，高級言語が使用できないことがあるが，新規に開発されたプロセッサを採用する場合，しばらくの期間は高級言語の開発が追いつかないことがある．このような場合でも，アセンブラの処理系は比較的短期間のうちに利用可能になる．新規のプロセッサを使用したシステムのハードウェア開発途中において，ハードウェアの正常な動作を確認したり，不具合を追求したりする場合，プロセッサの出力する信号が予測できるアセンブリ言語でのプログラミングが不可欠である．

以上5つの例にはあげなかったが，システムの性能を左右するBIOSやOSの開発，言語処理系の性能を左右するコンパイラとライブラリの開発などでアセンブリ言語が生き残っているケースが多い．

CPUを知る ◆ マイクロプロセッサに直接触れる

アセンブリ言語でプログラミングをはじめるには，何といってもまず使用するCPU（プロセッサ）のことを知らなければならない．ここでは，プログラミングを行ううえで必要なポイントをあげることにする．

リスト4　デバイスドライバの先頭部分

```
; デバイスヘッダ
;
        DD      -1              ; リンクポインタ
        DW      ATTRIBUTE       ; デバイスの属性
        DW      STRATEGY        ; ストラテジルーチンの入り口
        DW      INTERRUPT       ; インタラプトルーチンの入り口
        DB      'DEVNAM  '      ; デバイスの名前
;
; ここから実際のプログラムが始まる
;
COMMAND DD      1 DUP(?)
STRATEGY:
        MOV     WORD PTR COMMAND,BX
        MOV     WORD PTR COMMAND+2,ES
        RETF
;
```

リスト5　BCD加算を行う関数

```
void bcdadd(char *x, char *y, int n) {
        _asm {
                MOV     CX,n
                JCXZ    BA2
                PUSH    DS
                LDS     SI,x
                LES     DI,y
                CLC
        BA1:    MOV     AL,[SI]
                ADC     AL,ES:[DI]
                DAA
                MOV     [SI],AL
                INC     SI
                INC     DI
                LOOP    BA1
                POP     DS
        BA2:
        }
}
```

アセンブラ用語集
BIOS　Basic Input and Output System（基本入出力システム）の略で，システムコールで呼び出すことができる．

図1　8086のレジスタ構成

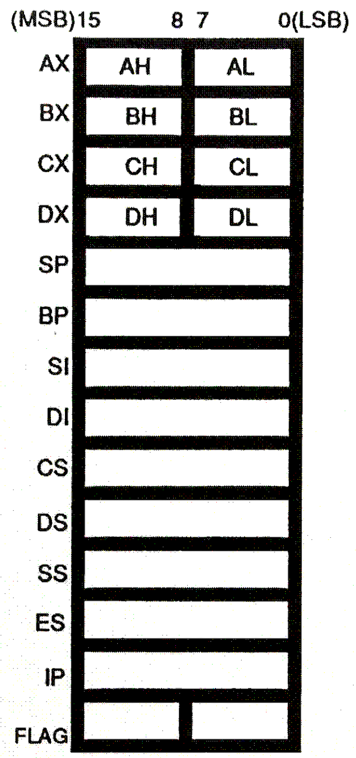
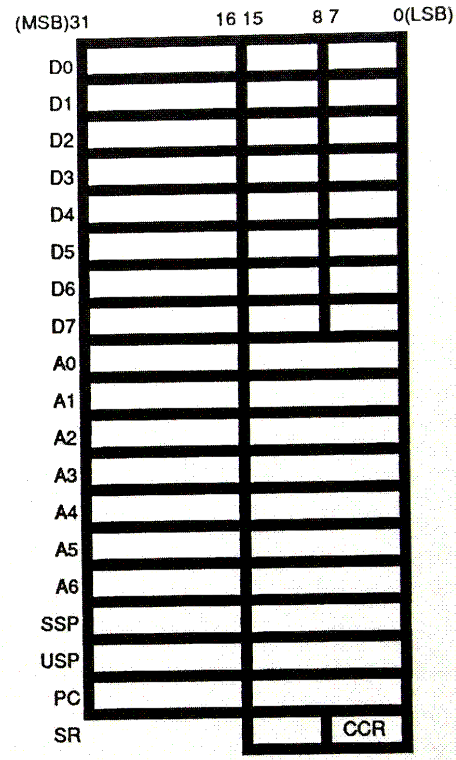


図2　68000のレジスタ構成



レジスタ

レジスタは，CPU内部にあるデータの一時記憶場所である．レジスタはメモリと同じように使用できるが，メモリとは異なりCPUの内部にあるので高速に利用できる．そこでアセンブラのプログラミングにおいては，このレジスタを上手に利用できるかどうかがプログラムの性能を左右することになる．

レジスタを上手に利用するためには，レジスタの構造と用途を理解する必要がある．レジスタの構造は，図1から3に示すようにCPUによってかなり異なっている．これらの図の最上段に書かれている数字はビット番号である．

8086の場合，すべてのレジスタは16ビットである．AXからDXまでのレジスタは8ビットに分割されて独立に機能する．これらのレジスタは8ビット単位でアクセス可能だが，たとえばAXとALレジスタは，名前は違っても別べつの場所ではないから注意しなければならない．8086では，アドレス空間は20ビット（1MB）であるにもかかわらず，プログラムカウンタ（8086ではインストラクションポインタという．IPのことである）も，スタックポインタ（SP）も，そのほかのアドレスを指定するために使用するレジスタ（BX，BP，SI，DI）も16ビットである．これを補うために，セグメントレジスタ（CS，DS，SS，ES）が存在する．セグメントレジスタの値は，16倍されてアドレスを指定するレジスタ（IP，SP，BX，BP，SI，DI）の値に加算され，20ビットのアドレスを作成する．IPにはCSが，SPにはSSが必ずペアで使用されるが，そのほかのレジスタには，いちおうデフォルトのセグメントレジスタが決まっているものの，CS，DS，SS，ESの各セグメントレジスタを指定することもできる．

68000の場合，SR（ステータスレジスタ，8086のFLAGに相当する）を除くすべてのレジスタは32ビットである．プログラムカウンタ（PC）も32ビットであるが，実際にはアドレス空間は24ビット（16MB）しかなく，上位8ビットは無視される．レジスタは，おもに演算に使用されるデータレジスタ（D0～D7）とアドレス指定に使用されるアドレスレジスタ（A0～A6，SSP，USP）に分かれている．データレジスタは8ビット（ビット0～7），

アセンブラ用語集
アキュムレータ　汎用レジスタの中でもAXレジスタをとくにアキュムレータという．算術，論理演算の中心となるレジスタ．

16ビット(ビット0～15),32ビット(ビット0～31)のアクセスが可能だが,8086とは異なりアクセスするビット数により位置が固定される.アドレスレジスタは16ビット(ビット0～15),32ビット(ビット0～31)のアクセスが可能であるが,これも位置が固定され,しかもアドレスレジスタに対して16ビットの操作を行うとアドレスレジスタに残る結果は必ず32ビットに符号拡張される.

68000には,スーパーバイザー状態とユーザー状態と呼ばれる2つの特権状態があり,それぞれの状態に対してSSPとUSPの2つのスタックポインタをもっている.ちなみに68000では,8086のようなセグメントレジスタはなく,16MBのアドレス空間を直接使用することができる.アセンブリ言語では,D0～D7をR0～R7,A0～A6をR8～R14,SSPとUSPを区別する必要のないところでは,A7,SPあるいはR15と表記することがある.

Z80の場合,汎用レジスタ(A,B,C,D,E,H,L)は基本的に8ビットである.しかし,特定の命令では,BとC,DとE,HとLの各レジスタを組み合わせて16ビットのレジスタとしても使用できる.AレジスタはPUSH,POP,EX命令でFレジスタ(8086のFLAGに相当する)とペアで操作されるが,そのほかの場合は8ビットとして使用される.汎用レジスタとFレジスタは2組あるが('のつくものとつかないもの),これらは同時には使用できず,EXあるいはEXX命令で切り換えて使用する.

オリジナルのZ80では,アドレス空間は16ビット(64KB)で,アドレスはすべて直接使用できる.ただし,各社から発売されている上位互換の機種では,特別なレジスタを設けてアドレス空間を増加させたものがある.IXとIYのレジスタは,16ビットのインデックスレジスタで8ビットの符号つきオフセットとともに,メモリのアドレスを指定するのに使用される.Iレジスタは,割り込みベクトルの位置の上位8ビットを指定するレジスタで,Rレジスタはダイナミック RAMのリフレッシュカウンタである.

フラグレジスタ

フラグレジスタは,プログラムからアクセス可能なレジスタの一種である.このレジスタの用途のひとつは,プログラムからCPUの状態を制御することである.もうひとつの用途は,CPUがひとつひとつの命令を実行するたびに,その結果の概略(たとえば,キャリーの有無,符号の正負など)を自動的に設定することである.そして設定された内容は,高級言語のif文に相当する条件ジャンプ命令によって調べられ,条件が満たされていればプログラムの流れを変えるのにしばしば利用される.表1～3は,各CPUのもつフラグレジスタ(68000ではSR,Z80ではFレジスタ)の内容を示したもの

図3　Z80のレジスタ構成

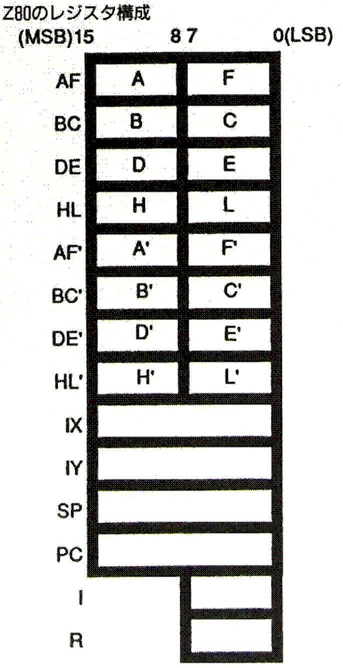


表1　8086の FLAG の内容

BIT	記号	意味・役割
11	O	オーバーフロー
10	D	ストリング命令の方向
9	I	外部割り込み許可
8	T	トレース
7	S	符号 (1 : MINUS)
6	Z	ゼロ
4	A	補助キャリー
2	P	パリティ (1 : EVEN)
0	C	キャリー

表2　68000のレジスタ構成

BIT	記号	意味・役割
15	T	トレースモード
13	S	特権状態
10	I2	外部割り込みマスクレベル
9	I1	
8	I0	
4	X	多精度演算用キャリー
3	N	符号 (1 : MINUS)
2	Z	ゼロ
1	V	オーバーフロー
0	C	キャリー

アセンブラ用語集
アドレッシングモード　扱うメモリアドレスの参照方法をいう.アドレスをレジスタを介さずオペランドで直接参照する場合を直接アドレッシングという.

である.

8086のフラグレジスタの内容のうち,下位バイトは8ビットのCPU8080/8085と同じである.ビット2とビット1を除けば,Z80とも同じである.表1の上から順番に簡単に説明しておこう.

表3　Z80のFの内容

BIT	記号	意味・役割
7	S	符号（1：MINUS)
6	Z	ゼロ
4	H	補助キャリー
2	P/V	パリティ・オーバーフロー
1	N	減算
0	C	キャリー

Oフラグは,演算の結果オーバーフローが発生すると1にセットされ,そうでなければ0にリセットされる.

Dフラグは,ストリング命令のアドレスを自動減少にするか,自動増加にするかをプログラムから設定して指定するビットである.1にセットすると自動減少になる.

Iフラグは,1にセットすると外部割り込みを許可する.割り込みが発生すると0にリセットされる.なおINT命令によるソフトウェア割り込みは,Iフラグの状態に無関係でいつでも実行できる.

Tフラグは,1にセットすると1命令の実行が終了するたびにトレース割り込みを発生する.割り込みが発生すると0にリセットされる.トレース割り込みはプログラムのデバッグに使用される.

Sフラグは,命令が実行された結果,符号ビットが1になるとセットされ,結果が負であることを示す.符号ビットが0である場合,0にリセットされ,結果が正であることを示す.

Zフラグは,命令の実行結果が0であると1にセットされ,そうでなければ0にリセットされる.

Aフラグは,補助キャリーと呼ばれ,BCD形式の演算に使用される.1バイト単位の計算にのみ有効で,ビット3からビット4への桁上げがあったとき1にセットされ,そうでなければ0にリセットされる.

Pフラグは,命令が実行された結果1のビットが偶数個であれば1にセットされ,奇数個であれば0にリセットされる.結果のパリティを表すビットである.

Cフラグは,算術演算やシフトなどの命令で桁あふれが発生したことを示すビットである.桁あふれが発生すれば1にセットされ,そうでなければ0にリセットされる.

68000のSRの内,上位バイトにあるT,S,I2～I0はスーパーバイザー状態でのみ操作できるCPU自身の状態を制御するフラグである.SRの下位バイトはCCRと呼ばれ,プログラムの実行結果の概要が設定されるフラグで常に操作可能である.

68000のT,N,Z,V,Cフラグは,それぞれ8086のT,S,Z,O,Cフラグに対応しているので説明は省略する.

SはCPUのリセットを含めたすべての例外発生時に1にセットされ,CPUはスーパーバイザー状態にされる.Sをプログラムで0に戻すとユーザー状態になる.

I2, I1, I0は3ビットの割り込みマスクレベルで,このビットで設定されているレベル以下の外部割り込みは禁止される.ただし,優先順位のもっとも高いレベル7の割り込みは禁止できない.

68000のCフラグは,データの移動命令でも変化するのが特徴である.ところが,このような変化を受けると多精度の演算がやりにくくなる.そこで,この問題を避けるため,算術演算とシフト系命令ではCフラグと同じ変化をし,そのほかの命令では変化しないXフラグが設定されている.68000の多精度演算を行う命令は,すべてこのXフラグを参照する.

Z80では,P/VフラグとNフラグを除いて8086のFLAGの下位バイトと同じである.

Z80のHフラグは,8086のAフラグに対応する.P/Vフラグは,算術演算系の命令のときオーバーフローを表す8086のOフラグと同様である.論理演算系の命令では,8086のPフラグと同じようにパリティを示す.

Nフラグは,減算と比較の命令で1にセットされ,加算系の命令で0にリセットされる.Hフラグと共同して,BCD演算に使用される.

データ構造

レジスタのデータ構造は,図1～3に示した通りである.メモリ上のデータ構造でとくに問題になるのは,多数のバイトをまとめて1つの数値あるいはポインタとして使用するときに,上位バイトから下位バイトをどのような順序で並べるかということである.各CPUのメモリ上のデータ構造を図4～6に示す.ただし,ここに示したデータ構造は,命令が直接扱う構造であり,効率を重視しないならプログラム

アセンブラ用語集
イミディエイトオペランド　オペランドとしてレジスタ参照ではなく,直接16ビットまたは8ビットの値を指定すること.

図4　8086のデータ構造

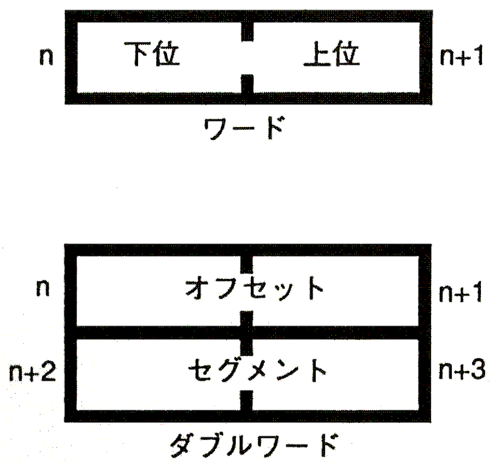


図5　68000のデータ構造

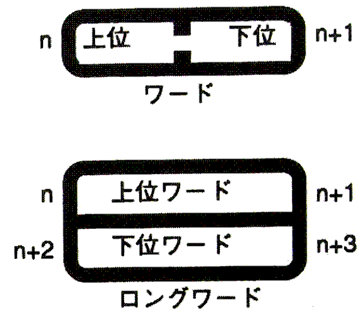
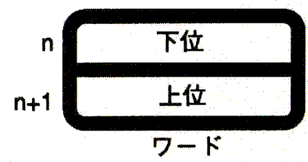


図6　Z80のデータ構造



を工夫することによりどのようなデータ構造でも扱えることはいうまでもない.

図4でnは，メモリの任意の番地を示している．メモリの番地上，ワード（2バイト）では下位バイトが先に，上位バイトがあとに置かれる．ダブルワード（4バイト）のデータも同様に下位ワードを先に，上位ワードをあとに置く．オフセットとセグメントからなるダブルワードのポインタでは，オフセットのワードを先に，セグメントのワードをあとに置く．nを偶数番地（ワード境界）に置くと一度に2バイトのデータがアクセスできるので実行速度を上げることができる.

68000の場合，ワード（2バイト）とロングワード（4バイト）をアクセスするためにはnが偶数番地でなければならない．8086とは上位と下位の配置が逆で，上位バイト，上位ワードを先に，下位バイト，下位ワードをあとに配置する．これより多精度のデータを扱うときも，これにしたがって最上位バイトを先に，最下位バイトをあとに配置しないと，68000の命令語の関係で操作がやりにくくなる．なお，バイト単位でアクセスする場合には，nは任意の番地でよい.

Z80の場合，命令語で直接扱える多精度のデータはワード（2バイト）だけである．8086と同様，下位バイトを先に，上位バイトをあとに配置する．nは任意の番地でよく，nがどこにあってもアクセス速度に影響は受けない.

アドレッシングモードと実効アドレス

アドレッシングモードとは，命令が操作の対象とするデータ（オペランド）の位置（実効アドレス）を指定する方法である．オペランドのない命令であっても，操作の対象は必ず存在する．そのような場合のオペランドは，インプライドオペランドと呼ばれる.

以下にあげるアドレッシングモードは，オペランドのある命令で使用される基本的なものである．例は，8086のアドレッシングモードだけをあげたが，他のCPUでもほぼ同じである.

命令によって使用できるアドレッシングモードはまちまちであるので，どのアドレッシングモードが使用できるかは命令ごとに覚えておかなければならない．傾向として，MOVなどのデータ移動に関するアドレッシングモードがいちばん複雑であり，ADD，ADC，SUB，SBBなど1群にできる命令では，同じアドレッシングモードをもっている.

①レジスタ

指定したレジスタの内容がオペランドとなる．たとえば，

アセンブラ用語集
インクルード　include疑似命令などでヘッダやソースファイルをプログラムに挿入すること。

```
MOV     AX, BX
ADD     AX, DX
```

などはすべてレジスタの内容がオペランドになる.

②イミーディエット（即値）

命令語に含まれるデータがオペランドになる. たとえば,

```
MOV     AX, OFFSET X
SUB     AX, 8
```

のOFFSET Xや8は, それらの表す数値がそのまま命令語に組み込まれ, その数値がオペランドとなる. 実効アドレスは命令語に含まれるデータのアドレスである.

③メモリ直接

メモリのアドレスを直接指定し, そのアドレスにあるメモリの内容がオペランドになる. 8086の場合, セグメントベースからのオフセットであることも, セグメントベースとオフセット両方のこともある. たとえば,

```
MOV     AL, Y
CALL    FAR PTR SHORI
```

のYは, DSレジスタをセグメントベースとするオフセットであり, FAR PTR SHORIはセグメントベースとオフセットの両方をオペランドとする. 実効アドレスは, 指定されたメモリのアドレスそのものである.

④メモリ間接

メモリ直接の場合と同様に, オペランドにメモリのアドレスを指定し, そのアドレスにあるメモリの内容がさらにメモリのアドレスとなり, そのメモリから得られたアドレスにあるメモリの内容がオペランドになる. 8086ではよい例がないが, たとえば,

```
INT     21H
```

の21Hはベクトル番号で, メモリアドレス0000:0084Hと同じものである. そして, この命令では, 0000:0084Hの内容であるセグメントベースとオフセットのアドレスから割り込み処理がはじまる. この場合の実効アドレスは, 0000:0084Hの内容である.

⑤レジスタ間接

レジスタの内容がそのままメモリアドレスになる. たとえば,

```
MOV     AX, [SI]
MUL     WORD PTR [DI]
```

の[SI]および[DI]がレジスタ間接オペランドである. この場合, SIレジスタの内容とDIレジスタの内容が, DSレジスタの内容をセグメントベースとするオフセットとなり, そのアドレスにあるメモリの内容が操作の対象となる. 8086では, [SI]および[DI]のほかに[BX]がある. 実効アドレスはレジスタの内容である.

レジスタ間接の変形として, レジスタ間接後自動増加, レジスタ間接前自動減少, レジスタ間接後自動減少というアドレッシングモードもある. たとえば,

```
PUSH    AX
POP     ES
```

のAXは, オペランドには直接現れないが, SPの内容のオフセットアドレスを2減少させたあと, SPの内容をアドレスとしてそのアドレスに格納される. これをレジスタ間接前自動減少という. また, POP命令のオペランドにあるESには, SPの内容をアドレスとしたメモリから取り出された内容が格納される. そしてそのあと, SPの内容は2増加する. これをレジスタ間接後自動増加という. 同様に, レジスタ間接後自動減少もストリング命令で使用される.

⑥インデックス修飾

レジスタの内容に, ディスプレイスメントと呼ばれる符号つきの数値を加えたものが実効アドレスとなる. 8086のディスプレイスメントは, 8ビットと16ビットの2種類がある. インデックス修飾は, テーブルの操作に便利なアドレッシングモードである. たとえば,

```
LEA     DI, X[SI]
MOV     ES, 2[DI]
```

のX[SI]や2[DI]がインデックス修飾のオペランドである. このアドレッシングモードで使用されるレジスタをインデックスレジスタと呼ぶ. 8086のインデックスレジスタとして使用できるレジスタには, SIとDIのほかにBXとBPがあるが, BXとBPはとくにベースレジスタと呼ばれている. それは, 次のようなインデックス修飾の変形に使用されるからである.

```
MOV     AX, 16[BP][SI]
```

16[BP][SI]の実効アドレスは, BPレジスタの内容とSIレジスタの内容とディスプレイスメントである16をすべて加えたものである. このアドレッシングモードでは, 2つのレジスタのうち, 一方はBXまたはBPでなければならず, 他方はSIまたはDIでなければならない.

⑦プログラムカウンタ相対

アセンブラ用語集
インストラクションポインタ　CPUが次に実行するべき命令が入っているアドレスが格納されている.

8086では，命令語の次のアドレスから何番地先か何番地あとかをオペランドとして指定するものである．これは，プログラムカウンタをインデックスレジスタとしたインデックス修飾と同じものと解釈できる．たとえば，

```
CALL    NEAR PTR SAGYO
JB      ERROR
```

のSAGYOやERRORは，メモリ直接と区別がつかないが，プログラムカウンタ相対のオペランドである．実効アドレスはラベルから計算されるディスプレイスメントにプログラムカウンタの値を加えたものである．このアドレッシングモードをもつ機械語は，アドレスに依存しない．

命令

アセンブリ言語で使用する命令は，ある程度動作を想像できるようなニーモニックと呼ばれるアルファベットの名前がついている．ニーモニックは，どのアセンブラでもだいたいCPUチップ製造メーカーの発表したニーモニックを採用しているが，アセンブラによっては多少差異のあることがある．また，機能の似通ったニーモニックを1つにまとめたり，1つの命令に2つ以上のニーモニックが存在する場合もある．表4は，各CPUのニーモニックを筆者なりに機能別に整理したものである．

それぞれの命令の詳細は，各CPUやアセンブラのマニュアルにゆずるとして，以下に命令を実際に使用する際，必要な注意事項をあげる．このほか，例外の発生条件を記憶しておけば，デバッグに便利なことがある．

①動作

動作を誤解なく素早く理解するためには，動作の略記法を読めるようにするとよい．たとえば，8086のCALL命令には次のような略記法で動作が説明されている．

```
SP = SP-2
[SP]= PC
PC = EA
```

②オペランドの数とアドレッシングモード

オペランドの数は命令によって違う．オペランドのない命令もあれば，オペランドを複数とる命令もある．オペランドを複数とる命令では，どのオペランドにどのアドレッシングモードが使用できるのかも注意しなければならない．

オペランドの数とアドレッシングモードを記憶するには，似通ったグループの命令ごとに整理するとよい．たとえば，8086の場合，ADC，ADDなどの算術演算に関する命令とAND，ORなどの論理演算に関する命令は同じオペランドをとる．

③影響を受けるフラグ

影響を受けるフラグもオペランドと同様グループ化して記憶すると覚えやすい．ただし，CPUの中にはニーモニックが同じでも，オペランドが異なるとフラグの変化が異なるものがあるので注意する必要がある．たとえば，Z80のINCやDEC命令はオペランドが16ビットか8ビットかによってフラグの変化が異なる．

ハードウェアを知る◆回路図からプログラムを作成する!?

アセンブラプログラミングに必要なハードウェアの知識の範囲は，プログラミングの対象によって大幅に異なる．極端な場合，ハードウェアの回路図からプログラムを作成する場合もある．このような場合には，ハードウェアの技術者と同じ知識を要求される．

極端な場合はさておいて，パソコン上でプログラミングをするための必要最低限と思われる例をプログラムを中心にあげてみよう．プログラム例は，PC-9801シリーズのハードウェアを想定している．

VRAMのアクセス

VRAMは主記憶の一部として，またCRTディスプレイに画面を出力するインターフェースのレジスタとしての役割をかね備えている．ハードウェアを制御するもっとも簡単な例として，VRAMを直接アクセスして画面を消去するサブルーチンを作成することにする．

画面を制御しているVRAMは，物理アドレスA0000H番地から2000ワードとA2000H番地から2000ワード

アセンブラ用語集
インデックスレジスタ　ストリング転送命令などで，メモリ中の配列やテーブルをアクセスしたときに参照されるレジスタ

(実際には上位バイトはない)である．全画面を消去するには，A0000H番地から2000ワードに0020Hを，A2000H番地から2000ワードに00E1Hを書込めばよい．これを8086のアセンブリ言語で書くとリスト6のようになる．

ポーリング

一般に入出力装置は，アセンブラのプログラムが実行される速度よりはるかに遅い．たとえば，ドットマトリックスプリンタでは，1秒間に数百文字印字できるだけである．何も対策をしないで，プリンタにアセンブラのプログラムから文字を出力しようとすると，1秒間に数万文字以上出力できるだろう．こうするとプリンタは処理が追いつかなくなり，どんどん文字を取りこぼすことは必至である．そこで，プリンタが1文字印字し終わったことをコンピュータに報告し，その報告を受けるまで文字をプリンタに送らないようにすることを考えなければならない．

このように，入出力機器の速度に合わせてコンピュータの操作を行うことを同期をとるという．VRAMのアクセスでは，システムのハードウェアが自動的に短期間CPUを待たせて同期をとってくれるのでプログラムで同期をとる必要はない．ところが，プリンタを相手にするように，長期間CPUを待たせると，CPUの動作に対する無駄が大きくなる．ハードウェアはこのような場合，プログラムに処理の仕方を任せるのである．プログラムで同期をとる第1の方法としてポーリングがある．

プリンタが文字を受けつけられるかどうかを示す信号線（BUSY）の状態を，アドレス42Hの入出力ポートのビット2で読みとることができるとする．このビットが1のとき，データはプリンタに受けつけられる．また，プリンタに送る文字のデータはアドレス40Hのポートへ書き込み，その直後にプリンタへデータを受け取らせるための信号線(STROBE)にパルスを送らなければならないものとする．この信号線には，アドレス46Hのポートに0EHを書き込み，適当な間隔(1マイクロ秒以上）をおいて同じポートに0FHを書き込むことでパルスが発生できる．このようすを図7に示す．

図7の信号を発生させるためのプロ

表4　機能別ニーモニックの一覧

	Z80	8086	68000
算術演算比較等	ADC ADD CP CPD CPDR CPI PIR DAA DEC INC NEG SBC SUB	AAA AAD AAM AAS ADC ADD CBW CMP CMPS CWD DAA DAS DEC DIV IDIV IMUL INC MUL NEG SBB SCAS SUB	ABCD ADD ADDA ADDI ADDQ ADDX CMP CMPA CMPI CMPM DIVS DIVU EXT MULS MULU NBCD NEG NEGX SBCD SUB SUBA SUBI SUBQ SUBX TST
論理	AND CPL OR XOR	AND NOT OR TEST XOR	AND ANDI EOR EORI NOT OR ORI
シフト等	BIT RES RL RLA RLC RLCA RLD RR RRA RRC RRCA RRD SET SLA SRA SRL	RCL RCR ROL ROR SAL SAR SHR	ASL ASR BCHG BCLR BSET BTST LSL LSR ROL ROR ROXL ROXR
移動・交換等	CCF DI EI EX EXX LD LDD LDDR LDI LDIR POP PUSH SCF	CLC CLD CLI CMC LAHF LDS LEA LODS MOV MOVS POP POPF PUSH PUSHF SAHF STC STD STI STOS XCHG XLAT	CLR EXG LEA MOVE MOVEA MOVEM MOVEP MOVEQ PEA Scc SWAP TAS
入出力	IN IND INDR INI INIR OTDR OTIR OUT OUTD OUTI	IN OUT	（入出力はメモリ空間に配置）
分岐	CALL DJNZ JP JR RET RETI RETN RST	CALL INT INTO IRET Jcc JMP LOOP LOOPZ LOOPNZ RET	Bcc BRA BSR DBcc JMP JSR RTE RTR RTS TRAP TRAPV
その他	HALT IM NOP	CS: DS: ES: ESC HLT LOCK NOP REPZ REPNZ SS: WAIT	CHK LINK NOP RESET STOP UNLK

リスト6　画面消去サブルーチン

```
;
; レジスタをスタックに待避する
;
CLEAR:  PUSH    AX
        PUSH    CX
        PUSH    DI
        PUSH    ES
;
; A000:0000Hから2000ワード分
; 0020Hをメモリに格納する
;
        MOV     AX,0A000H
        MOV     ES,AX
        MOV     DI,0000H
        MOV     CX,2000
;
; レジスタをスタックから回復する
;
        POP     ES
        POP     DI
        POP     CX
        POP     AX
        RET
```

アセンブラ用語集
インデント　行の先頭にタブや空白を入れること．プログラミングスタイルのひとつでソースファイルでプログラムの流れを判断しやすくする．

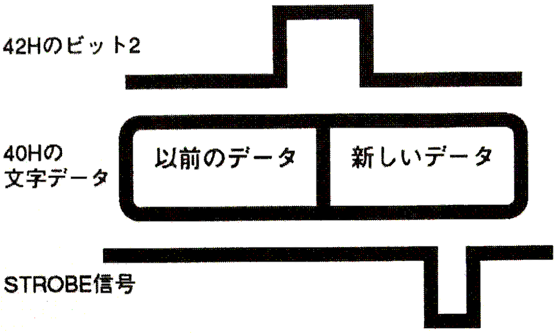


図7　プリンタを制御する信号

リスト7　プリンタへ1文字出力する

```
PRINT:  PUSH    AX
;
; プリンタが文字を受付可能になる
; までポーリングを行う
;
PRINT1: IN      AL,42H
        AND     AL,04H
        JZ      PRINT1
;
; 文字のデータを出力する
;
        POP     AX
        OUT     40H,AL
        JMP     SHORT $+2
;
; STROBE信号線にパルスを送る
;
        MOV     AL,0EH
        OUT     46H,AL
        JMP     SHORT $+2
        MOV     AL,0FH
        OUT     46H,AL
        RET
```

グラムを，8086のアセンブリ言語で書くとリスト7のようになる．リスト7のプログラムは，ALレジスタに入っている1文字を出力するサブルーチンの形で書かれている．

割り込みを使用する

前述のプリンタとの同期では，ポーリングという手法を用いたが，これはCPUがプリンタの準備ができるまで何もしないで待っているという単純な手法であった．これでは，CPU時間の無駄である．そこで，CPUを待たせない方法を紹介しよう．

リスト8は，10ミリ秒ごとのタイマ割り込みを使用して，何らかの実行時間を測るために使用できるプログラムである．

プログラムは，4つのサブルーチンからできあがっている．サブルーチンTOPNは，タイマのLSIと割り込みコントローラを初期化し10ミリ秒ごとの割り込みを開始させる．このサブルーチンは，時間の計測をはじめる前に必ず1度呼び出さなければならない．

サブルーチンTINTは，割り込み処理ルーチンと呼ばれる特殊なルーチンである．CPUは，タイマ割り込みが発生すると通常のプログラムを強制的に中断して，ベクトル番号8 (0000:0020H) のアドレスに格納されているポインタの指す割り込み処理ルーチンを呼び出す．サブルーチンTOPNが，このアドレスにTINTへのポインタを格納しているので，タイマ割り込みが発生するとTINTが呼び出される．TINTは，割り込みコントローラに割り込み処理が終了したことを知らせ，メモリ上の4バイトのカウンタTCNTをカウントアップする．割り込みは，10ミリ秒ごとに発生するので，TCNTは10ミリ秒ごとに＋1される．

TINTルーチンが終了すると，通常のプログラムが中断された位置へ何事もなかったように戻る．このようすを図8に示す．

通常のプログラムが時間を知りたいときは，サブルーチンTGETを呼び出す．サブルーチンTGETはTCNTの値，すなわち10ミリ秒単位の時間をAXレジスタとDXレジスタに格納して通常のプログラムへ返す．このようにして，CPUはタイマをポーリングすることなしに，通常のプログラムを実行しながら時間を計測できる．

サブルーチンTCLSは，割り込みコントローラのあと始末をしてタイマ割り込みを禁止し，割り込みベクトルを元に戻す．通常のプログラムは時間計測が終了したなら，TCLSを1回呼び出す必要がある．

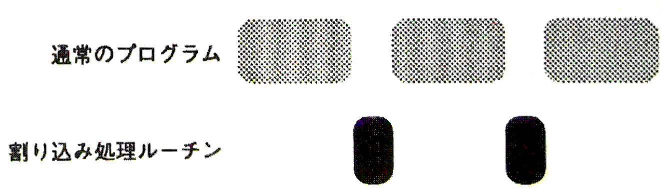


図8　割り込み処理ルーチンの処理状態

アセンブラ用語集
インラインアセンブラ　C言語などほかの言語のソースファイル中に直接アセンブラソースを記入すること．

リスト8　時間計測サブルーチン

```
TOPN:   PUSH    DS
;
; 割り込みベクトルの設定
;
        XOR     AX,AX
        MOV     DS,AX
        MOV     SI,0020H
        CLI
        MOV     AX,[SI]
        MOV     CS:SAVE,AX
        MOV     AX,2[SI]
        MOV     CS:SAVE+2,AX
        MOV     AX,OFFSET TINT
        MOV     [SI],AX
        MOV     2[SI],CS
        STI
        POP     DS
        XOR     AX,AX
        MOV     CS:TCNT,AX
        MOV     CS:TCNT+2,AX
;
; タイマLSIの初期化
;
        MOV     AL,34H
        OUT     77H,AL
        JMP     SHORT $+2
        MOV     AX,19968
        OUT     71H,AL
        JMP     SHORT $+2
        MOV     AL,AH
        OUT     71H,AL
;
; 割り込みコントローラの初期化
;
        IN      AL,02H
        AND     AL,0FEH
        JMP     SHORT $+2
        OUT     02H,AL
        RET
;
TINT:   PUSH    AX
;
; 割り込みコントローラに割り込み処理
; 終了を通知する
;
        MOV     AL,60H
        OUT     00H,AL
;
; TCNTを+1する
;
        ADD     WORD PTR CS:TCNT,1
        ADC     WORD PTR CS:TCNT+2,0
        POP     AX
        IRET
;
TGET:   CLI
;
; TCNTの値をAXレジスタと
; DXレジスタに格納する
;
        MOV     AX,CS:TCNT
        MOV     DX,CS:TCNT+2
        STI
        RET
;
; タイマ割り込みを禁止する
;
TCLS:   IN      AL,02H
        OR      AL,01H
        JMP     SHORT $+2
        OUT     02H,AL
;
; 割り込みベクトルを復元する
;
        PUSH    DS
        XOR     AX,AX
        MOV     DS,AX
        MOV     SI,0020H
        CLI
        MOV     AX,CS:SAVE
        MOV     [SI],AX
        MOV     AX,CS:SAVE+2
        MOV     2[SI],AX
        STI
        POP     DS
        RET
;
SAVE    DW      2 DUP(?)
TCNT    DW      2 DUP(?)
```

OSを知る ◆ お行儀のいいプログラミングを

アセンブリ言語は，OSを直接呼び出す言語でもある．高級言語ではライブラリにかくれているので，OSを強く意識しなければならないことは少ないが，アセンブリ言語のプログラマは，ターゲットのOSを知らないと作成したプログラムを実行させることもできない．そこで，OSとアセンブラの関係について少し触れることにする．

OSは，コンピュータシステムの運用を円滑に進めるために資源の管理を行うものである．資源とは，CPU，メモリ，入出力，プログラム，データなどである．これはつまり，プログラムが使用する資源の使用法をOSの管理手法に限定することにより標準化することである．

MS-DOSを例にとると，MS-DOSで動作するプログラムは次のようなプログラミング上の規制を受ける．

①プログラム起動時のインターフェース

プログラムで最初に実行される命令の位置，設定されているレジスタの種類とその値，プログラムに割り当てられているメモリの量，コマンドラインの情報の受け取り方，環境変数を取得する方法などが規定される．

②資源の使用法

資源を利用するには，MS-DOSの機能を決められた方法で呼び出さなければならない．ドライブやファイルの命名法など，MS-DOSの機能を呼び出す際に指定するパラメータにも制限がある．

このような規制を受けるメリットは，規制を厳格に守る限り，MS-DOSの動作する機種すべてで実行できるプログラムを作成できる点である．さらに機種に依存した煩雑な入出力ルーチンから解放されるばかりでなく，プログラムで作成したデータファイルに対しても互換性をもたせることができ

アセンブラ用語集
オーバーフローフラグ　演算の結果，オーバーフローが生じたときにセットされるフラグ．

る．たとえば，リスト9にあるプログラムはMS-DOSの動作する機種ならどの機種でも，「Programmer's Page」というメッセージをコンソールディスプレイに表示してくれる．

リスト9 メッセージを表示するプログラム

```
MSDOS   EQU     21H
OUTSTR  EQU     09H
TERMP   EQU     4CH
;
TEXT    SEGMENT
        ASSUME  CS:TEXT,DS:TEXT
;
PRINT:  MOV     AX,CS
        MOV     DS,AX
        MOV     DX,OFFSET TEXT:MSG
        MOV     AH,OUTSTR
        INT     MSDOS
        MOV     AH,TERMP
        MOV     AL,0
        INT     MSDOS
;
MSG     DB      'Programmer''s Page$'
;
TEXT    ENDS
;
STACK   SEGMENT STACK
        DB      256 DUP(?)
STACK   ENDS
;
        END     PRINT
```

しかし，問題もある．MS-DOSの管理する範囲は標準的な資源に限られているから，特殊な入出力装置やMS-DOS以外のシステムで作成したファイルなどを扱おうとすると，BIOSの直接呼び出しや独自の入出力管理を行うといった，いわゆる“行儀の悪いプログラム”を作成しがちである．とくに，アセンブリ言語のプログラマは，筆者も含めてこの誘惑に打ち勝つことは困難なようである．

アセンブリ言語というのは，高級言語に比べるとスマートな言語でも楽な言語でもない．むしろ，地味な努力を必要とする言語である．それゆえ，初心者には取りつきにくい言語であろう．

しかし，多少の時間をかけてある程度のポイントを自分のものにしてしまえば，あとは他の高級言語と同じようにそれほど困難はないものである．苦労した以上に，コンピュータシステムの可能性をより多く自分のものにできる．アセンブラの楽しみを味わっていただきたいと望むしだいである．

アセンブラによるROM化プログラミング

アセンブラによるプログラミングは，プログラムの構成の自由度が大きいため，プログラムコードを書き換えたり，プログラムコードとプログラムコードの間にデータ領域を設けたりすることができる．ところが，ROM化されるプログラムではROMとRAMの2種類のメモリの使用区分を明確にしなければならない．ROMはCPUからライトアクセスができないし，RAMは電源を切ると内容が蒸発するためである．原則として，次のようにセグメント分けしてコーディングするとスッキリする．

- **ROMに配置する部分**
  - プログラムコード（書き換えは御法度）
  - 書き換えられないデータ（定数）
  - 初期化に使用するデータ
- **RAMに配置する部分**
  - 初期化の必要なデータ領域（書き換えられる）
  - 初期化不要のデータ領域（作業記憶やスタックなど）

上記のセグメントというのは，8086系のCPUがハードウェアでサポートしている物理的なセグメントというだけの意味ではない．市販のリロケータブルなアセンブラでは，物理的なセグメントをもたないCPUに対しても，たいてい論理的なセグメント分けができるようになっている．たとえば，M80のCSEGとDSEG，CP/M68K付属アセンブラの.TEXT，.DATA，.BSSといった疑似命令がこれに相当する．これらの疑似命令を使用してセグメント分けをし，リンク時にそれぞれのセグメントの開始アドレスをリンカに指定してやれば，ROMのバイナリイメージができあがる．リンカによっては，直接ROMライター形式のファイルに変換して出力してくれるものもある．

MASMとMS-LINKの組み合わせでも，やっかいではあるがROM化プログラミングは可能である．MASMでは任意の名前の多数のセグメントが使用できるので，ROMとRAMに配置する部分をさらに目的に応じて細かくセグメント分けすることができる．このとき，MS-LINKのセグメントの結合規則に従いセグメントを記述する順序，境界整列，属性，グループ化，クラス分けに注意する．ROMとRAMのオフセットのつじつまを合わせるために，セグメントとセグメントの間に領域を確保するだけのダミーセグメントを挿入することも必要になることがある．

このほか，セグメントアドレスはつねに再配置の対象になるので，セグメントアドレスの参照には定数をレジスタにロードして参照するなどの工夫がいる．このようにしてコーディングしたソースをMASMとMS-LINKで.EXEファイルに落とし，SYMDEBかDEBUGでメモリにロードしてROMの部分をファイルにセーブすればバイナリイメージができあがる．

アセンブラによるROM化プログラミングは，以上のような注意と工夫をすれば，それほど困難なものではない．しかし，能率や作業の容易さを重視するなら，ROM化に対応したアセンブラとリンカを使いたいものである．（秋山登）

アセンブラ用語集
オーバーヘッド　演算などが実行されない時間のこと．サブルーチンはマクロ参照よりサイズは小さくなるがオーバーヘッドが大きい．

ASSEMBLER

2

CPUアーキテクチャとマシン語

86系からRISCまでを概観する

宮川誠一（Mt.Trail／mix）

マシン語／アセンブラ言語は，それぞれのCPUアーキテクチャに密着した，CPUを直接操作する命令群からなるプログラミング言語である．現在使われているおもなCPUの発展の歴史とアーキテクチャを解説し，それぞれのCPUにおけるアセンブラプログラミングの特徴を述べる．

ここでは，ふだんよく耳にするCPUを取りあげてその特徴と系譜を紹介したいと思います．

まずは，身近なパソコンのCPUをとりあげます．パソコンのCPUというとインテル社の80x86系です．ちょっと前までは日本電気のパソコンは独自CPUであるV30を使っていましたが，最近ではまた80x86系に戻っています．この80x86系について，8086から80486までその特徴と違いを眺めてみましょう．

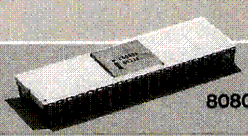
次に，80x86系が64KBのメモリの壁をひきずっている間に，いち早く32ビット時代を切り開いたモトローラの680x0系をとりあげます．680x0系CPUは，広いメモリ空間と高級言語との親和性から，いっきにワークステーションという分野を築きあげてしまいました．さらに，このパソコンとミニコンの間を埋める世界を実現したワークステーションでは，より早いマシンの要求からRISC技術を利用したCPUを採用しはじめました．現在では，ワークステーションの最上位クラスのほとんどがRISC CPUを用いています．このRISC CPUについても少し見ておきましょう．

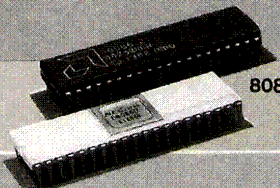
表1に，代表的なCPUが発表された年をまとめてみました．この年表から，1988年，1989年に出てきたCPUが非常に多いことがわかります．また，この年表を作る過程で人間との対話環境や計算環境に使われるCPUの流れと，組み込み用に使われるCPUの流れが重なりあって流れているのがわかりました．前者は高速化，高機能化をめざし，後者は周辺I/Oの取り込みをめざしています．CPUの多様化の流れはまちがいなく加速されているようです．

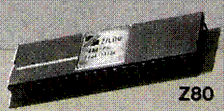
本題に入る前にひとこと断っておきますが，私はどのCPUがよいとか悪いとかをいうつもりはありません．私自身はCPUはすべて好きですし，すべてのCPUはその生きる場所がある

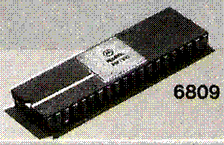
表1　CPU年表

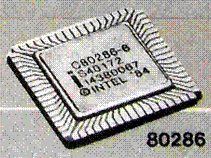
メーカー	1971	1972	1973	1974	1975	1976	1977	1978
Intel	4004	8008		8080				8086/88
Motrola					6800			
MOS Technologie					6502			
Zilog						Z-80		Z-8000
Sun Microsystems								
MIPS Computer								
NEC								
TRON								


8080


8086


Z80


6809


80286

アセンブラ用語集
オフセット　ある基準（セグメント）からの相対的な位置を示すのに使われる．

からこそ生まれてきたのだと思っています．適材適所，うまく生かしてやるのがわれわれの仕事ではないでしょうか．

80x86系

パソコンの標準CPU

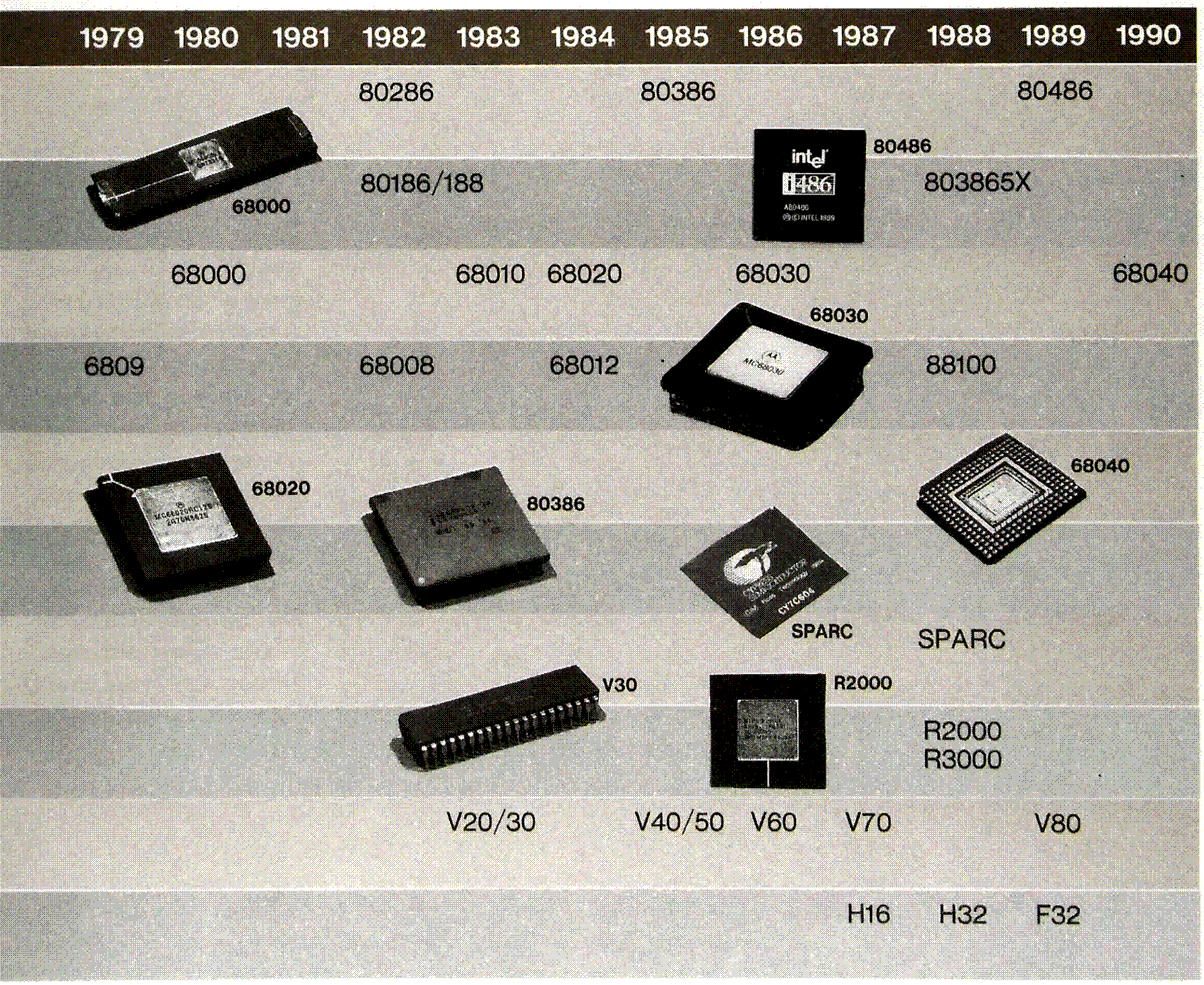
8086の系譜

8086は，その前の8ビット時代を継承するCPUとして登場しました．8ビット時代は，8080（インテル）から8085（インテル）そして8ビット時代の覇者Z80(ザイログ)と変化し，パーソナルコンピュータの黎明期を形作りました．

インテルは，8080で8ビット時代の先駆けとなりましたが，ザイログのZ80には遅れをとる結果となりました．モトローラも8ビット時代に6800から6809というCPUを出し，ある程度の健闘をしたのですが，パソコン分野ではCP/Mというオペレーティングシステムの前に敗退する結果となりました．現在のマイクロコンピュータの両巨頭は，ともに最初の戦いではちょっと失敗をしたわけです．

さて，急激なマイクロコンピュータの進化にともない，8ビット時代から16ビット時代への変化のときがきました．この時代は，いまから考えるとパ

写真提供：インテルジャパン／日本モトローラ／日本サイプレスセミコンワクタ／ミップスコンピュータジャパン
協力：小松伸行



アセンブラ用語集
オフセットアドレス　実アドレスの下位16ビットの値で，4ビットシフトさせたセグメントアドレスとの和が実アドレスとなる．

ソコンの戦国時代でした．ミニコンや汎用機の世界の人からは玩具のように見られていた８ビットパソコンが，16ビット時代を迎え仕事で使えるパソコンへと成長するとともに，パソコン世界の覇権争いが熾烈に行われました．

CP/M全盛ではありましたが，Appleなどもがんばっていました．この時代から独自路線で，いまでもがんばっているのはAppleだけですね．この戦いに終止符を打ったのは，IBMでした．IBMが，現在のMS-DOSの原型となるCP/MもどきのMS-DOSをかかげてパソコン分野に乗り出してきたのです．そのころのIBMは，いま以上に強大な勢力をコンピュータ業界にふるっていましたし，IBMのもつソフトの力には神聖な力が感じられる時代でした．

こうして，CP/M文化からMS-DOS文化への移行とともにIBMが選択した8088/8086というCPUが世界のパソコンの標準CPUの地位を築いたのです．また，８ビット時代のパソコン文化を支えた8080やZ80などとの互換性がとりやすかったことも，80x86系CPUが標準となった大きな要因のひとつでした．そして，この上位互換性という命題が以後のIntelのCPUアーキテクチャの十字架となるのです．

以後，パソコンは8086から80286へ，そして現在の80386の時代へと順次変化していきます．オペレーティングシステムはずっとMS-DOSできましたが，MS-DOSはハッキリとその方向性としてUNIX文化をお手本としてきました．

"仕事で使えるパソコン"という位置づけが要求する機能を追い求めたソフトウェアは，ユーザーインターフェースの改善，ファイルシステムの充実，通信環境の取り込み，データベースの充実などに取り組み，8086のメモリ空間をまたたくまに狭く感じさせてしまいました．

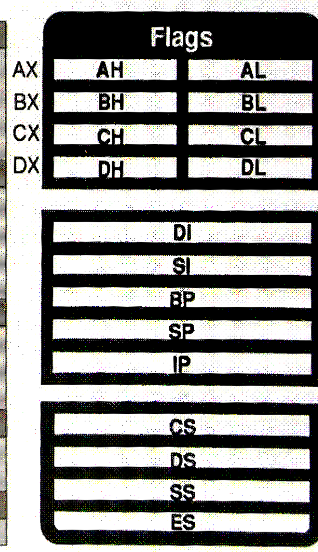
同時代の68000が，最初から８ビットとの互換性を捨て32ビット時代をめざしたのと裏腹に，8086は上位互換性を保つことを至上命令として16倍のアドレス空間をもつ80286へと変身しました．しかし，その互換性とプロテクトモードの難解さのため，80286は不幸にも高速の8086としてしか利用されなかったのではないでしょうか．

十字架は重たくのしかかり，80286から導入されたプロテクトモードが花咲くのは80386になってからでした．メモリの壁がなくなり，時代もやっとプロテクトモードを使いこなせるようになり，OS/2やWindows 3.0などの出現で，いま，別の世界を開こうとしています．新しいアーキテクチャが広く使いこなされるようになるまでには，４～５年はかかるのです．

新しいOSへの移行の最大の問題は，やはりソフトウェアの上位互換性ですが，最近ではUNIX文化に端を発するC言語文化のおかげで，CPUの違いの大部分は隠されるようになってきました．それでも，OSやデバイスドライバ，高速化を要求される部分などでは，CPUに密着したアセンブラが必要とされることは変わりません．また，逆説的ですがコンパイラへの要求が高まるほど，コンパイラの作成者にはCPUを使い切ることが要求されますし，コンパイラでしか書けない会社よりは，アセンブラでクリティカルポイントをチューニングできる会社のほうが競争力が強くなります．つまり，

表２　8086のレジスタ

セグメントレジスタ		
CS	コードセグメント	
DS	データセグメント	
ES	拡張データセグメント	
SS	スタックセグメント	
汎用レジスタ		
AX	汎用＋乗除計算用	(Accumulator)
BX	汎用＋ベースアドレス	(Base)
CX	汎用＋カウンタ	(Counter)
DX	汎用＋ポートアドレス＋乗徐計算用	(Double word)
ポインタレジスタ		
IP	命令（インストラクション）ポインタ	
SP	スタックポインタ	
BP	ベースポインタ	
インデックスレジスタ		
SI	（元）ソースインデックス	
DI	（先）デスティネーションインデックス	
フラグレジスタ		
FLAGS（キャリー，ゼロ，符号，パリティ，など）		



汎用レジスタは16ビットの上位と下位を８ビットずつに分けても利用できます．

AX：（AH，AL）
BX：（BH，BL）
CX：（CH，CL）
DX：（DH，DL）

Hのつくほうが上位（High）でLのつくほうが下位（Low）です．

アセンブラ用語集
オプティマイズ　最適化と訳す．むだなコードを省きプログラムサイズを小さく，または実行速度を速くするためにプログラムを加工すること．

コンパイラ文化が大きくなればなるほど，アセンブラ（CPU）を知っていることは強力な武器になるのです．

8086の特徴

●レジスタセット

8086には，表2のような14個の16ビットレジスタがあります．これは，8086の状態を完全に指定するには，28（14×2）バイトの情報があればよいことを意味します．

なお，最初のIBM PCで使われた8088というCPUは，データバスが8ビットということ以外はほぼ8086と同じと考えてよいCPUです．

8086のレジスタは，その使用方法がある程度最初から決められています．とくに80x86系の悪名高きセグメントレジスタは，オフセット用のレジスタまたは数値と組み合わせて，実際に読み書きするアドレスを決定するのに使われます．別の見方をすると，1 MBの空間中にアクセス用の4つの窓を作るのに使われるということです．

名前から予測できる通り，CSは次に実行する命令コードのアドレスをIPとの組み合わせで決定するコード用の窓です．SSは，SPとの組み合わせでスタック位置を決定するスタック用の窓です．DSは標準的に使われるデータのセグメントを示し，ESはDSだけではたりない部分を補うのに使われるデータ用の窓です．

汎用レジスタとBP，SI，DIは，かけ算と割り算以外の演算が可能です．かけ算と割り算には，AXとDXが固定的に使用されることになっています．4バイト（Double word）の数値となる場合，ダブルレジスタ＝DXが上位16ビット，アキュムレータ＝AXが下位16ビットとなります．

汎用レジスタのところで「＋何とか」と書いてあるのは，そのレジスタには特別に決められた使用方法があることを示しています．BXはベースアドレス，つまり配列などの先頭アドレスに使われることを想定しています．CXはループカウンタとして，DXは乗除算の上位用のほかに16ビットのポートアドレスを指定するのに使われます．

BXだけがレジスタ間接アドレッシングに使え，AX，CX，DXではレジスタ間接アドレス指定ができません．

ポインタレジスタは，データへのポインタに使われます．もちろん，IPとSPはすでに述べたように，インストラクションポインタとスタックポインタ専用です．BPはベースポインタと呼ばれていますが，これは高級言語などでパラメータ受け渡しや自動変数領域に多用されるスタックフレームの基準（ベース）アドレスを指定するのに使用されます．

インデックスレジスタは，名前の通りにデータのインデックスとして使用されることを想定したレジスタです．

これらのレジスタは，アドレスを生成するために使われるときにはいくつか組み合わせて使用することができます．

基本の構成は，

ベース＋インデックス＋オフセット

つまり，

[BP | BX]＋[SI | DI]＋[数値]

となります．ここでベースの部分にBPを使用した場合には，デフォルトのセグメントレジスタがSSになり，スタック上のデータを操作するのに向いています．そのほかの場合には，デフォルトのセグメントレジスタはDSになります．デフォルトとは，「何も指定しない場合」という意味です．

なお，この式で［］で囲まれている部分は，最低1つあればほかは省略可能です．BP | BXのように | で区切られている部分はどちらかを選択します．ここで，SPがないことに注意してください．スタックポインタは，スタック上のデータをアクセスするのには使えません．スタック上のデータのアクセスには，SPをBPに代入しBPを使います．

デフォルトのセグメントの例外が1つだけあります．それはストリング命令と呼ばれる命令で，この中ではDIはESと組み合わせることになってい

リスト1　アドレス生成の例

```
0000                      data    segment
0000  0064[0000]          table   dw       100 dup(0)
00C8                      data    ends

                          point   struc
0000  0000                  x     dw       ?
0002  0000                  y     dw       ?
0004                      point   ends

0000  8B 84 0000 R                mov      ax,table[si]
0004  8B 84 0000 R                mov      ax,[table+si]
0008  8B 84 0000 R                mov      ax,[si].table
000C  8B 84 0000 R                mov      ax,[si]+table

0000  89 40 02                    mov      [bx+si+2],ax
0003  89 40 02                    mov      [bx][si+2],ax
0006  89 40 02                    mov      [bx][si]+2,ax
0009  89 40 02                    mov      [bx][si].y,ax
```

アセンブラ用語集
オペランド　ほとんどの命令は，ふたつのデータ間で処理を行う．この命令の実行対象とするものをオペランドという．

ます．ストリング命令についてはあとで説明します．

アセンブラ(MASM)でのかんたんな表現例をリスト1にあげておきます．

●8ビットCPUからの継承

Z80や8080/8085，およびその互換CPUを知っている人には，8086のレジスタの名前はほとんど同じような名前なので，非常にすんなりと受け入れられるでしょう．また，命令セットそのものも，ほぼ同じようなものがそろえられているので，8ビット時代に作られたアセンブラのソースを8086に移し変えるのは非常にかんたんに行えます．

私自身もはじめて8086の世界に入るときデバッグモニタを移植しましたが，ほとんど機械的に置き換えるだけで基本の部分は動きました．あとは拡張された部分を追加して最初のデバッグモニタを完成させました．

次節のアドレス空間のところでもう少し詳しく述べますが，8086のセグメントレジスタをすべて同じ値にして64KB以内でプログラミングするような場合を考えると，8086はほぼ8ビットCPUのレジスタが16ビットになったような感覚でプログラミングできてしまいます．つまり，64KBに収まってしまうようなプログラムを作る場合には，ほとんどセグメントを意識する必要がありません．

この64KBに収まるプログラムが，MS-DOSのCOM形式のプログラムです．このCOM形式は，CP/M時代のソフトをMS-DOSに移植するには最適でした．ちなみに，CP/M時代の実行形式の拡張子がCOMでしたので，MS-DOSでもこの呼び方を継承しています．

●アドレス空間

8086の最大の特徴は，アドレス空間の複雑さでしょう．物理的にCPUそのものから出ているアドレスは20ビット(1MB)です．しかし，8086には16ビットレジスタしかありません．どうしても4ビット不足してしまいます．

そこで考えられたのが，16ビット＋16ビット→20ビットという計算式です．どのような8086の入門書にものっているように，

セグメント×16＋オフセット

で，実際のアドレスを作り出します．

レジスタの説明のところで紹介したいくつかのレジスタと数値をたし算することによって作り出される値も，最終的には16ビットで計算されることに注意してください．このことは，オフセット部分の式の計算結果が溢れても捨てられて若いアドレスのほうへまわり込むことを意味しています(図1)．積極的に考えるなら，負の値を加えると若いアドレスのほうへ動くとも考えられます．つまり，16ビットの符号付計算と符号なし計算の関係です．

そうやって計算された16ビットのオフセット値とセグメント値からアド

図1　まわり込むようす(1)

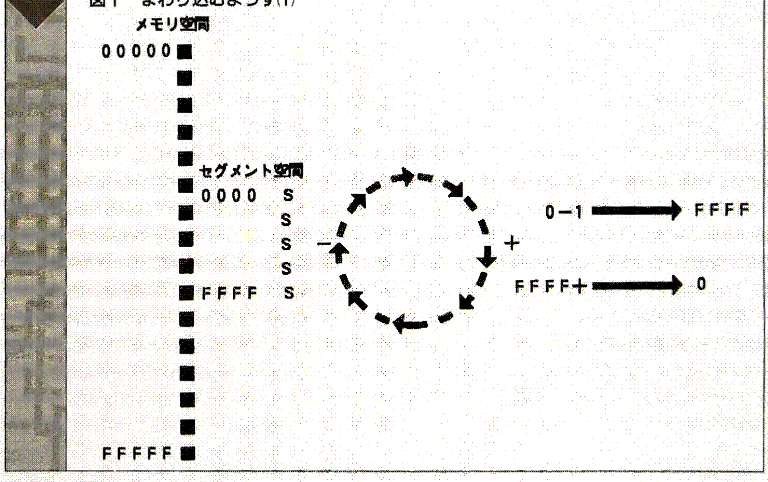
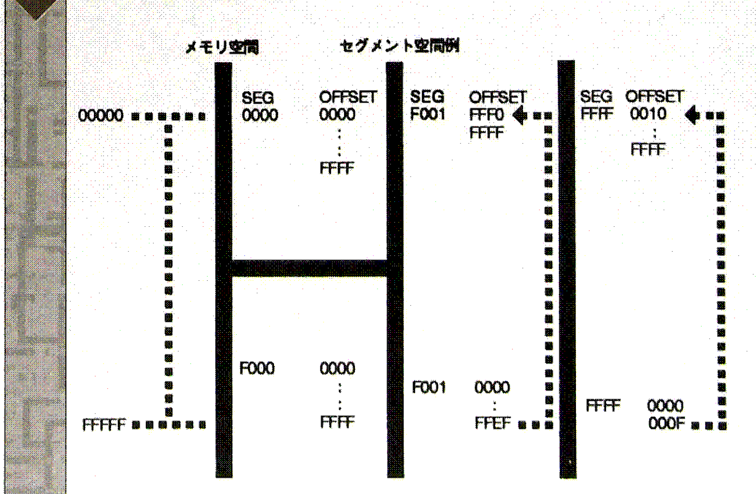


図2　まわり込むようす(2)



アセンブラ用語集
カウンタ　CXレジスタはおもに繰り返し命令やシフト・ローテート命令の実行回数などのカウンタとして使われる．

レスが計算されるのです．そして，このセグメントとオフセットのたし算のときには20ビットで切られます．ですから，20ビットを越えるアドレスは0番地のほうへまわり込むことになります．結果として，F001HからFFFFHまでのセグメント値をもつ場合には0番地へのまわり込みが発生します（図2）．symdebなどのデバッガで，この間のセグメントを指定してダンプしてみるとかんたんにわかります．

また，8086では同じアドレスを表現するセグメントとオフセットの組み合わせは，4096通り（10000H÷10H）あります．

この1セグメント16ビット（64KB）の方式は，上位互換性を保ちながら拡張するためにはしかたなかったのかもしれませんが，現在まで続いている大きな障壁となってしまいました．根本的な問題は，レジスタ幅がアドレス空間幅よりも小さいというところにあります．

アドレス変換はCPUの中で行われ，CPUから外に出る部分では実アドレスであることに注意してください．このため，ハード的にはとくに内部のアドレス生成方法を意識する必要はありません．同じアドレスの表現が4096通りあっても，ハードウェア的に出力されるアドレスは1つです．

●I/O空間

8086には，メモリ空間とは別にI/O用のポートアドレス空間が64KBあります．この空間には，IN／OUT命令などの特殊な命令を使用することでアクセスすることができます．この点，68000系はメモリマップドI/Oと呼ばれるメモリ空間の一部をI/Oに使用する方式をとっているので異なっています（注：80x86でもメモリマップI/Oは行うことができます．）．

8086のI/O空間では，最初の0から255までのI/Oアドレス空間にはI/O命令に直接アクセスアドレスを書くことができますが，それ以上のI/OアドレスにはすべてDXレジスタにI/Oアドレスを設定してDXレジスタを指定してしかアクセスできません．

●セグメント，64KBの壁

8086をけなすときに，いつもいわれるのが64KBの壁の話です．アドレス空間のところで説明したように，セグメントで示されるベースアドレスから64KB（16ビット）の範囲しか直接にはアクセスできないことから出てくる問題です．

もちろん，セグメントレジスタの値を変えれば1MB空間の任意のアドレスをアクセスできますから，64KBしかアクセスできないわけではありません．

通常のプログラミングではほとんど問題にならない場合が多いのですが，最近のように高解像度のグラフィックデータや非常に点数の多い表計算をしようとすると，途端にプログラムがめんどうになってしまうのです．

しかも，8086にはセグメントレジスタが4つしかありません．このことは，最大4つの64KB空間しか，セグメントの切り替えなしにはアクセスできないことを意味しています．さらに，命令用（CS）とスタック用（SS）は通常決まってしまいますから，ユーザーが自由に使えるのはDSとESの2つだけということになります．

そのため，64KBを超えるデータを統一的に扱うためには，DSとESを切り替えながらアクセスすることになります．ただし，この壁の問題が現れるのは，64KBを超えるデータを扱う場合だけということを強調しておきます．

それでも，セグメントを意識しないといけない場合が多い8086のアドレッシングは，プログラミングに余分な注意を必要とするということに違いはありませんし，64KBの壁のためにワークステーションの世界へ80x86系CPUが出遅れたことはまちがいありません．

8ビット時代からの互換性の鎖が大きく影響し，結局は80386まで待たなければこのメモリの壁の崩壊は起らなかったのです．ただし，このメモリの壁の崩壊はOSの移行も伴いますから，まだまだいろいろな問題が起るでしょう．

●NEAR/FAR, SMALL/LARGE

8086のセグメントの問題は，サブルーチンコールなどでNEARかFARかという違いを意識しなければいけない結果を生みます．つまり，実行アドレスを切り替えるのにセグメントの変更を伴う（FAR）か，伴わない（NEAR）かという違いです．この違いは，呼び出し先から戻るためにスタック上にセーブするリターンアドレスがセグメントを含むか含まないかという点にも出ます．

8086では，できるだけNEARですませられるところはNEARですませてしまうことがプログラムサイズやスタックサイズを小さくし，実行時間を早くするコツです．ですから，できるだけ大きな単位でコードセグメントをまとめ，内部はNEARにし，セグメント間の呼び出しをできるだけ少なくするように設計したほうがよいことになります．

NEARとFARをまちがえて呼び出すと確実に暴走しますから注意が必要です．CPU自体には，実行時に呼び

アセンブラ用語集
間接アドレッシング　直接アドレッシングに対して，メモリアドレスの指定をレジスタを通じて行うものを間接アドレッシングという．

出し先がNEARであるかFARであるかを判定することが不可能なのですからしようがないですね.

呼び出しと同じように，データ領域などのアクセス時にも，データの存在するセグメントが複数ある場合には，セグメントレジスタの切り替えを行いながらデータにアクセスする必要があります．データが全体で64KB以内ならそんな必要はまったくないので，データアクセス用のセグメントレジスタの切り替えは不要になります.

データもコードも64KB以内であれば，非常にかんたんになることになりますね. このような設定をSMALLモデルと呼びます. LARGEモデルは，逆にコードもデータも複数のセグメントからなっているようなモデルです. この中間状態を表すMEDIUMとかCOMPACTとかありますが，このへんはコンパイラなどによって多少呼び方が異なります.

●予約領域

8086は，割り込み発生時に割り込みベクタテーブルといわれる飛び先を記録した領域を参照して必要な処理に分岐します. この割り込みベクタ領域は，メモリの0番地から3FFH番地までに固定的に割り付けられています．ほかへ移すことはできません.

また，その中の先頭から32個のベクタ（4バイト×32）領域はIntel予約となっています．CPU自身が検出した異常用のベクタが定義されています. しかし，MS-DOSなどのBIOSにこの領域を使っているものがあるのです. そのため，インテルは80386を作成したときに，コプロセッサ例外の仕様をわざわざ対応せざるをえなくなってしまいました.

もう1か所予約領域があります．それは，FFFF0HからFFFFFHまでの領域です．ここは，リセットスタートのコード領域として予約されています．8086は，電源投入時やリセットスタート時にはCS = FFFF, IP =0となります．ですから，通常はこの場所に初期化プログラムへのジャンプ命令が書き込まれています.

●ちょっと違った流れの80186

8086のあとに出た80186は，ちょっとメインの流れとは異なったCPUです．8086よりも少し強化された命令語をもつと同時に，周辺LSIを取り込んだ形になっています．最近ではよく見られる形式ですが，1チップマイコンなどと同じようにローコスト・高密度の組込み分野などに向けた製品になっています．今回は，パソコンやワークステーションのCPUを重点的に見ていますのであまり80186には深入りしないことにします.

●高速化のための工夫

8086のアーキテクチャの中で高速化に貢献しそうな特徴を紹介してみましょう．まずはストリング命令.

ストリング命令は，1バイトまたは2バイト単位のデータの並び（配列）に対しての処理を高速化するための命令です．ストリング命令には，転送，比較，探索，ロード，ストアの5種類があります．これらの命令は，1回の実行後，処理対象へのポインタを自動的に更新する機能があるため，連続して命令を行うリピートプレフィックスと組み合わせて使用すると非常に効果があります.

このストリング命令は，ガチガチにレジスタが決められています．ソース側はDS:SI，デスティネーション側はES:DIに固定です．唯一，ソース側のセグメントだけは，セグメントプレフィックス（セグメント切り替え）命令をつけることで別のセグメントレジスタに切り替えることができます.

ストリング命令をうまく利用すると，プログラムを高速化することができます.

もう1つの特徴は，プリフェッチキューでしょう．8086には，6バイトのプリフェッチキュー（命令の先読みバッファ）があります．このため，ハードウェア的にアクセス速度の遅いメモリを使用していても，CPUが計算をしている間にメモリから次の命令を先に取り出しておけるためメモリの遅さをかなり吸収することが可能です．結果として，システムを作るのに安い(遅い）メモリを使っても，実行速度が落ちにくいということがいえます.

プリフェッチキューのサイズですが，この6バイトは通常出現する8086のもっとも長い命令語の長さに一致しています．一般的には命令語はもっと短いので，キューの中には2～3つの命令が先読みされていることになり，最悪でも1命令は入っていることになります. これに対して68000では，最長10バイトくらいの命令があるにもかかわらず，プリフェッチキューは4バイトしかありません．このへんのバランスは，インテルのほうがよいようです.

●命令語の注意

ここでひとつひとつの命令語を取りあげる余裕はありませんので，8086に特徴的な部分と注意しなければならない点についてだけ説明します.

まず，8086系はワード（2バイト）データやダブルワード（4バイト）データなどをメモリと読み書きする場合に，下位バイトから順に割り付けるという点です．これは，一般にリトルエ

アセンブラ用語集
疑似命令　アセンブルされてマシンコードになるのではなく，セグメントの指定や生成するコードの種類などアセンブラに対しての指示を疑似命令という.

ンディアンと呼ばれています．この逆が68000系などのビッグエンディアンと呼ばれる方法で，上位バイトから順に割り付けます．このため，68000系と8086系では2バイト以上のサイズの変数のメモリ上の配置が逆順になります．

もうひとつ，8086では条件ジャンプ命令では，命令の位置から＋130または-126の範囲でしか飛べないことです．これはちょっと短くて，すぐ届かなくなることが発生してめんどうなんですが，条件を逆にしたり無条件ジャンプと組み合わせたりして避けるしかありません．

80286

私にとって80286は中途半端なCPUです．その中途半端さゆえに，80286は高速8086として使用されることがいちばん多かったのではないでしょうか．しかし，パソコン用にはもっとも大量に使用されているCPUでしょう．

80386の出現と技術者がプロテクトモードになれてきたことが重なって，最近やっとその実力を発揮しはじめているようにも見えます．

プロテクトモードという言葉が出ましたが，80286には8086とまったく同じような動作をするリアルモードと，新しく導入されたプロテクトモードの2つの動作モードがあります．リアルモードで動かしているかぎり，80286を使っても1MB空間しか使えませんが，8086よりも2～3倍高速に8086用のプログラムを実行することができます．プロテクトモードでは，16MBの実アドレス空間を最大1ギガバイトの論理空間に割り付けて使うことができます．

「リアル」vs「プロテクト」は，CPUのメモリ空間の扱い方を表現したものです．つまり，セグメントとオフセットを数値的にたしたものがそのまま実（リアル）アドレスを表すことになるモードがリアルモードで，セグメントの考え方を拡張して各セグメントレジスタで示される空間の属性を管理する（プロテクト）機能を追加したモードがプロテクトモードです．

ですから，プロテクトモードではセグメントレジスタに入っている内容とオフセットをたし算してもアドレスにはなりませんし，セグメントレジスタに入っている内容がメモリを参照するのには使えない場合さえあります．

パソコンの需要が多い80286の最大の設計上の問題点は，リアルモードからプロテクトモードへ切り替えることはできても，プロテクトモードからリアルモードへ切り替えることがソフトウェア的にできないことでしょう．そのため，プロテクトモードとリアルモードを行き来するOSは，いったんリセットするという荒技を使用します．このため動作が遅くなってしまいますし，設計上もきれいになりません．パソコン以外の世界では別に問題にはなりませんが，MS-DOSには大きな足かせです．

●新しいレジスタ

一般ユーザーから見た場合には，80286には新しいレジスタの追加はありません．ここで追加されたのは，表3のシステム管理用のレジスタ群だけです．また，これらのレジスタはプロテクトモードのシステムプログラムを書く場合にしか必要ありません．リアルモードだけしか使用しない場合には，まるで存在しないかのように無視してかまいません．

あとは，内部的にこれらのレジスタを補佐するキャッシュ用のレジスタがありますが，ユーザーには直接関係ありませんのでここでは省きます．ただし，CPUから外部へのメモリアクセスを少なくするのには貢献していますし，切り替えタイミングが重要な処理を書く場合にはこれらを理解する必要があります．

MSWは，プロテクトモードへの移行やコプロセッサの状態などを管理するレジスタです．MSWのPEビットを1にするとプロテクトモードに移るのですが逆はできません．

TRはタスクレジスタと呼ばれるレジスタです．80286のプロテクトモードでは，CPU自身の中に複数の処理を同時に行うことをサポートする機能が入っているのです．この機能を使用すれば，かんたんに複数の処理を切り替えることができるようになります．TRには，現在実行中の処理（タスク）の情報が保持されています．このような処理の単位のことをタスクと呼んで

表3　80286で追加されたレジスタ

MSW	マシンステータスワード
TR	タスクレジスタ
LDTR	ローカルディスクリプタテーブルレジスタ
GDTR	グローバルディスクリプタテーブルレジスタ
IDTR	割り込みディスクリプタテーブルレジスタ

アセンブラ用語集
キャリーフラグ　演算命令実行後，加減算による桁上がりや桁下がりが生じたときにセットされるフラグ．

います．OSのタスクとはちょっと違った，もっと低いレベルの管理単位です．

LDTR, GDTR, IDTRは同じような内容を指しています．それはディスクリプタテーブルといわれるものですが，内容についてはもう少しあとでお話します．

●リアルモード

リアルモードについては，8086と同じですのでここではかんたんに説明します．

リアルモードでの8086からの機能強化は，次のような点です．

- ブロックI/O命令の追加
- スタックフレーム操作命令の追加
- プッシュ／ポップ(全汎用レジスタ，イミディエート値)
- 配列の境界チェック

●プロテクトモードって何だ

プロテクトモードについて少し紹介しておきます．

80286から取り入れられたプロテクトモードでは，セグメントレジスタにもっている値は実際のセグメントの位置情報ではなくなります．この値はセレクタとも呼ばれ，ディスクリプタテーブルのインデックスになります．

ディスクリプタテーブルとは，そのセレクタで選択される空間がどのような属性をもっているかを書いてあるディスクリプタ（8バイト）が並んでいるテーブルです．ディスクリプタの中に，この空間はメモリセグメントで実メモリの何番地から何バイトあるかなどの情報が書かれています．

ディスクリプタはメモリセグメントだけではなく，タスクだとかゲートだとかの情報を保持している場合もあります．メモリセグメント以外を指定して読み書きの命令を出すとエラーとして割り込みが発生します．

メモリセグメントの場合でも，範囲のチェックやその領域にアクセスできる権利をもっているかどうかをCPUが自動的に調べてくれます．そのための情報もディスクリプタの中に入っているのです．

このようにメモリへアクセスする場合でも，セグメントレジスタで示されるディスクリプタから管理情報を調べ，正常な要求かどうかを検査し，アドレスを変換して実際のメモリの読み書きが行われます．ただし，セグメントレジスタに対応するディスクリプタの内容は毎回メモリから読み込まれるわけではなく，セグメントレジスタへ値が代入されるときに自動的に内部のキャッシュ領域へ読み込まれますから，以後の操作のときにはディスクリプタを読むためのメモリアクセスは発生しません．

ディスクリプタテーブルには，CPUに1個ですべてのタスクに共通なグローバルディスクリプタテーブルと，各タスクごとに独立にもてるローカルディスクリプタテーブルがあります．また，8086での割り込みベクタに相当する割り込みや例外処理を制御するためのディスクリプタテーブルもあります．これらの現在値を記録しているのが，GDTR，LDTR，IDTRです．

割り込みもディスクリプタを経由して，実際の割り込みハンドラへ実行が移ることに注意してください．割り込みディスクリプタテーブルは，8086の割り込みベクタと異なり0番地になくてもかまいません．

プロテクトモードには，4レベルの特権モードがあり，実行できる命令の制限やサブルーチンなどの呼び出し制

図3　ディスクリプタの構造

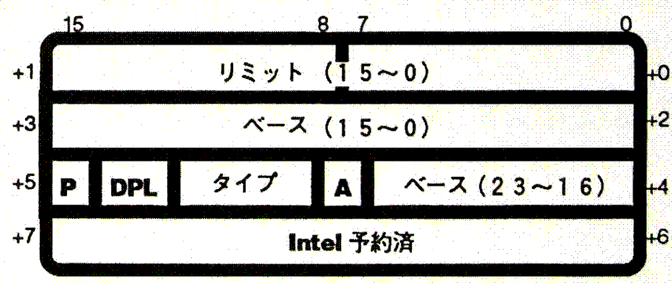
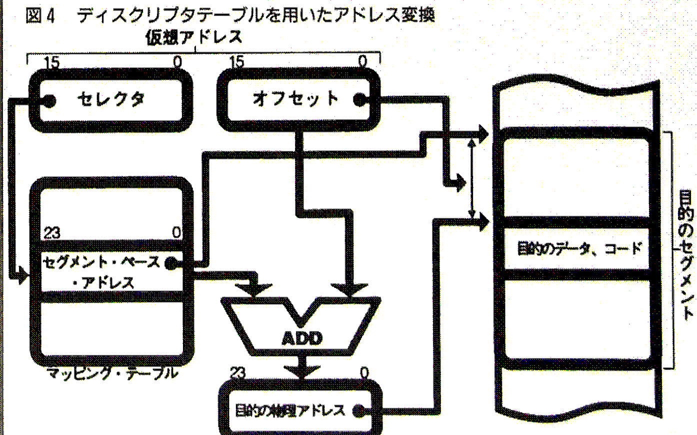


図4　ディスクリプタテーブルを用いたアドレス変換



アセンブラ用語集
クオッドワード　64ビット（8バイト）で表せる値をクオッドワードといい，倍精度浮動小数点などを表すのに使われる．

限を行うことができます．たとえば，割り込み禁止命令などのように危険な命令は使えないようにするとか，OS用のサブルーチンを正しくない手順で呼び出せないようにするとか，アクセス権のないデータを読み書きできなくすることなどができます．このように，プロテクトモードでのセキュリティはかなり厳密に守ることができるようになっています．なお，各タスクの各レベルごとに独立にスタックを割り付けることができます．

●残念なのは64KBの壁を破れなかったこと

このように80286からプロテクトモードが用意され，CPU自身の中にメモリ管理機構が組み込まれたのですが，レジスタの変更がなかったために80286は16ビットレジスタしかもっていませんでした．

その結果，オフセット部を16ビット以上にする実用的な手段がなく，64KBの壁をひきずり続けることになりました．プロテクトモードで実メモリが16MBまで可能といっても，1つのメモリセグメント内では64KBしか許されないのです．しかも，プロテクトモードにしてしまうとセグメント値を調整して次のアドレスをアクセスするなどという方法が許されなくなってしまうため，64KB以上の空間をアクセスするにはめんどうな新しい技術が必要になってしまったのです．

はじめて出現したプロテクトモードだったことを考慮しても，やはりレジスタの拡張がなかったこと，つまり中途半端な拡張だったことがプロテクトモードがほとんど使用されなかった原因のように思います．

●80386よりも速い?

ちょっと悪いところを書きすぎたかな？　と反省してというわけではないのですが，80386は80286よりもさらに管理機能が増えているためか？　同じクロックでの16ビットコードの処理速度は80286のほうが80386よりも速いという話もあるようです．

32ビットを使ったプログラムがまだそんなに多くない現状では，「うーむ？」と考えさせられるものがあります．

80386

やっとメモリの壁が取り除かれた80x86系のCPUです．80386の出現でプロテクトモードも実用的になり，パソコンがさらに飛躍する土台を築けるようになりました．

プロテクトモード時にも，8086のプログラムを同時に同じCPU上で実行可能とするための仮想86機構やCPU

表4　80386で追加されたレジスタ

セグメントレジスタ		
FS	データ用	
GS	データ用	
汎用レジスタのレジスタ幅拡大（16→32）		
AX	→	EAX
BX	→	EBX
CX	→	ECX
DX	→	EDX
デバッグレジスタ		
DR0	ブレークポイントアドレス0	
DR1	ブレークポイントアドレス1	
DR2	ブレークポイントアドレス2	
DR3	ブレークポイントアドレス3	
DR4	予約	
DR5	予約	
DR6	ブレークポイントステータス	
DR7	ブレークポイント制御	
テストレジスタ		
TR6	テスト制御	
TR7	テストステータス	
制御レジスタ		
CR0	モード制御	
CR1	予約	
CR2	ページフォルトリニアアドレス	
CR3	ページディレクトリベース	

表5　アドレス指定に使用できるレジスタの組み合わせ

ベース＋	（インデックス	＊スケール）＋数値
EAX	EAX	1
EBX	EBX	2
ECX	ECX	4
EDX	EDX	8
ESP	―――	
EBP	EBP	
ESI	ESI	
EDI	EDI	

自身の故障診断機能，さらにデバッグ機能の充実などが目をひきます．

さらにレジスタの使用方法の制限などが緩和され，セグメントレジスタの追加なども行われ使いやすくなっています．また，命令の実行もパイプライン化され，プリフェッチキューも16バイトに増えています．

●新しいレジスタ

セグメントレジスタの追加で，いままで自由に使えたDS, ESに加えて，FS, GSが使えるようになったため，セグメントレジスタの入れ換えが大幅に少なくなりました．プロテクトモードでは，セグメントレジスタの入れ換えは内部のディスクリプタキャッシュの更新が発生するため，かなり重たい処理ですから，必然といえばいえるのでしょうが，プログラミングが楽になることはまちがいありません．

汎用レジスタがすべて32ビットになり，さらに汎用レジスタの使い方がかなり自由になりました．このことがプログラミングを非常にやりやすくしてくれます．アドレスの指定に使えるレジスタの組み合わせは表5のようになります．

表5からわかるように，EAXでアドレスを指定することもできれば，スケールをかけ算して実際のアドレスを作るように指定することもできます．もしもあなたが80386のアセンブラを使う機会があれば，この差はすぐに実感できるでしょう．

●プロテクトモードが便利になった

80386になってレジスタが32ビットに拡張されたので，実アドレス空間のサイズとレジスタサイズが同じになりました．それにともなって，各メモリセグメントのサイズも4GBまで可能になっています．これで，大きなデータも1つのセグメント内で問題なく扱えます．

また，プロテクトモードとリアルモードの間は，ソフト的に自由に行き来できるようになったので，変な技を使わなくてもよくなってすっきりしています．

●仮想 86 モード

80386から追加された仮想86モード（V86)が，いちいちリアルモードに戻らなくても，MS-DOSのような8086のプログラムをプロテクトモードのままで実行することを可能にしました．

この仮想86モードのおかげで，ソフト的なラムディスクドライバやキャッシュドライバ，EMSドライバなどがたくさん出現するようになりました．

ただし，仮想86モードで動いているプログラムはさまざまな監視を受けます．また，ハードウェア割り込みなどが発生した場合には，プロテクトモードですのでプロテクトモード側の割り込みディスクリプタテーブルにもとづいて割り込み処理が実行されます．実際には，仮想86のタスクへ渡すべき割り込みはソフト的に処理を行ってやる必要があります．

このように仮想86を動かすにはいろいろなオーバーヘッドが増加しますので，実行速度的にはかなりの低下が見られます．それでも，この機能によって実現できることのメリットのほうが十分に大きい場合が多いようです．

●メモリ管理機構

メモリ管理機構もよくなりました．80286ではセグメント単位でしか管理できなかったのが，セグメントでもページ単位でも管理できるようになったので，余分なハードウェアの追加なしで仮想記憶を実現できます．

バイト単位の管理の場合，1セグメント1MBまで可能で，ページ（4KB）単位の管理の場合1セグメント4GBまでの管理が可能です．

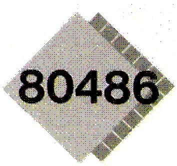
管理できる最大の論理空間は64TB（テラバイト）にもなります．

●デバッグレジスタ

80386のように複雑なCPUになってくると，開発用のICEなどの環境が整備され安心して使用できるようになるまでには長い時間がかかります．また，かなり高価なツールを購入しなければならなくなります．

しかし，80386にはデバッグレジスタがあります．4点のブレークポイントをCPU自身が監視してくれるのですから頼もしいかぎりです．これを使用すればROMの領域にもブレークポイントがかけられますし，データのアクセスなどでもブレークをかけることが可能です．

このデバッグレジスタを使用したデバッガさえ用意すれば，ほとんどのデバッグ処理は要求を満たすことができるでしょう．現実に私自身もデバッガを書きましたが，快調に動いています．おかげで，まだ開発装置が追いつかない80486のシステムもデバッグすることができました．これからのCPUにはぜひ欲しい機能だと痛感しています．



80486は，見かけ上のソフト的なスペックとしては80386+80387の世界とほとんど変わりません．80387が内蔵された80386と思ってプログラミン

アセンブラ用語集
サインフラグ　演算の結果で，最上位ビットが1のときセットされ，0のときリセットされる．正負の判断に使われるフラグ．

グしてかまいません.

そのほかの大きな目玉はキャッシュメモリの内蔵です. また, このキャッシュ機能は複数のCPUをつなぎ合わせたシステムを構成する場合についても考慮されたもので, キャッシュ内に入っているデータのもとをほかのCPUが更新しても,論理的な矛盾が起こらないようにすることができます.

80486を使用すると, 浮動小数点を多用するプログラムなどは劇的に早くなります. また, キャッシュをONにするだけでいっきに2.5倍のスピードになったプログラムもありました. ドライストーンテストなどでは, 最初の読み込み以外, CPUのほかにはいっさいアクセスに行くことなく終わってしまいます.

80486には, おもしろい命令が追加されています. それはBSWAPという命令で, 32ビットデータのビッグエンディアンとリトルエンディアンの形式変換を行ってくれるものです. つまり, バイト順序をそっくり入れ換えてくれる命令です. これでMOTOROLA系のCPUとデータをやり取りしても, かんたんにバイト順序を入れ換えることができます.

680x0系 ◆ ワークステーションからMacまで

680x0系の系譜

80x86系が8ビットの資産を継承するために上位互換性を重視したのに対し, モトローラは8ビット時代の鎖をばっさりと切り捨てて, 32ビット時代のCPUという発想で68000を発表しました.

680x0系は, その中では上位互換性をとれるように拡張されています. 80x86系にくらべてレジスタ数も多く, 多彩なアドレッシング方法をサポートしています.

最初に発表された68000のバスを8ビットにした68008, 68000に仮想記憶対応機能を追加した68010, さらにアドレス空間を2GBに拡張した68012, 32ビット化された68020, メモリ管理機構とキャッシュを内蔵した68030と変化し, 長いトラブルのあと, 最近やっと浮動小数点演算機能を内蔵した68040が出荷されはじめたようです.

680x0系はワークステーションなどの世界だけではなく, レーザープリンタの制御用などの組込み分野にも広く使用されています. 現在では680x0系のCPUをコアに使用し, 周辺LSIを取り込んだものもあります.

●特徴

何といっても, 680x0系の特徴はメモリ空間が単一でわかりやすいことと, レジスタ数が多いことでしょう. 80x86をやっているときにわずらわしく感じるセグメントレジスタがないのですから, もうそれだけで選択の理由になるかもしれません.

最初から上を想定してのアーキテクチャだったので, レジスタは32ビットでしかもデータ用8個とアドレス用8個があり, 無用なレジスタのプッシュ/ポップを減らすことが可能です. ただし, 恐がりな人のプログラムでは, レジスタを全部セーブしたりすることになり, かえって増えてしまう場合もあるようですが.

アドレッシングモードがたくさんあり, ソース側とデスティネーション側で同じようなアドレッシングモードが使えるので, 80x86系ではできないメモリ↔メモリの転送なども1つの命令でできるのは便利です.

68000には, ユーザーモードとスーパーバイザーモードという動作モードがあります. 名前の通りユーザーモードは一般的なプログラムの実行時のモードで, スーパーバイザーモードはOSや割り込み処理のプログラムの実行時に使用されるモードです. 2つのモード用のスタックは独立して割り当てられます. モードを分けることによって, 危険な命令を一般プログラムから実行できなくすることができるため, 安全性が高くなります.

680x0の命令語はすべてワード(2バイト)単位となっています. ですから, 奇数バイト数の命令はありません. さらに, 奇数アドレスから命令を実行しようとすると, 例外割り込みが発生してしまいます.

●割り込み処理

80x86系から680x0系に移った人がとても驚くのは, 680x0には割り込みの禁止/許可という命令がないことでしょう. 680x0系の割り込みはレベル管理(0−7)され, 現在の割り込みレベル以上の割り込み要求はいつでも受けつけられます. 割り込みが受けつけられると割り込みベクタテーブルから実行アドレスが取り出され, 割り込

アセンブラ用語集
サブルーチン　サブルーチンはプログラム中に繰り返し使われる命令ブロックをまとめ, 本体からCALL命令で呼び出す.

んだ割り込みレベルで実行を開始します．ですから，すべてのマスク可能な割り込みを禁止したい場合には，現在の割り込みレベルの設定を最高優先度（レベル7）にすることで行います．

ただし，この方法には見落としやすい落し穴があります．それは割り込み処理の先頭で割り込み禁止（レベル7）にしても，一度には1つの割り込みしか発生しないようにすることはできないことです．割り込みが受けつけられてから，そのレベル設定の命令が実行されるまでに，さらに優先度の高い割り込みが発生すると受けつけられてしまいます．ですから，最大8回の割り込みに耐えられることが必要です．このような微妙な点が問題になるのは，OSのスケジューラでも書く人かOSのない組込み用プログラムを作成している人くらいでしょうが，80x86系とは割り込み関係の常識が異なることを知っておいてください．68000系では，割り込まれた処理のスタックにセーブされた割り込みレベルの値を検査することによってしか，戻り先が割り込み処理であるかそうでないかを確実に知ることはできません．

また，本来はCPUが検出するエラーなどによって，CPUの内部的に発生する割り込みを例外と呼び，外部要因の割り込みを割り込みと呼びます．しかし，ここではすべて割り込みという言葉で通しています．

68000

●レジスタ構成

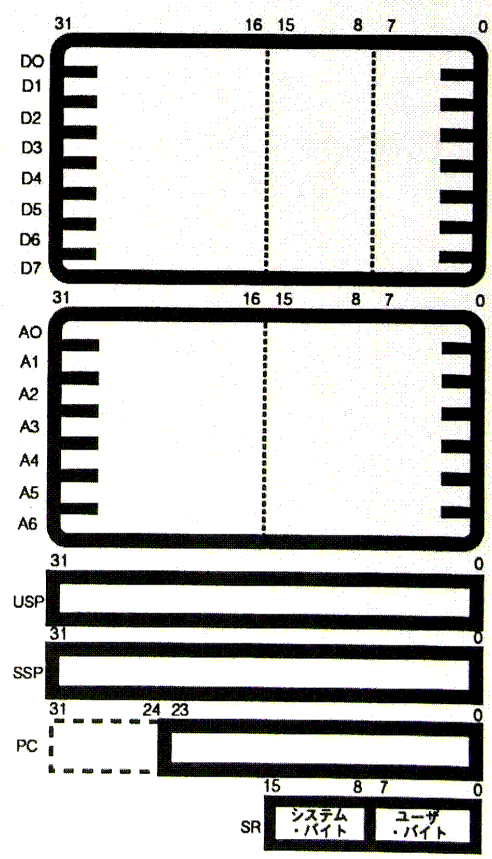
データレジスタは，おもに演算用に使用されます．32ビットなのでロングワードデータも扱えます．80x86のように，レジスタの一部をいくつかの独立したレジスタのように扱うことはできません．16ビットや8ビットのデータとしては下位側が使用されます．

アドレスレジスタは，おもにポインタとして使用されます．アドレスレジスタは，16または32ビットデータしか扱えません．16ビット値を代入しても32ビットに符号拡張されます．

表6　68000のレジスタ

データレジスタ
D0 から D7 の 8 本
アドレスレジスタ
A0 から A7 の 8 本 ただし A7 はスタック用でスーパーバイザ用とユーザー用が用意されている． どちらが A7 になるかは動作時のモードで決定される．
プログラムカウンタ
PC
ステータスレジスタ
SR

SR レジスタを除き32ビットレジスタです．



アセンブラ用語集
システムコール　ソフトウェア割り込みのこと．CPUはこの命令を受け付けると，割り込みポインタアドレスを参照して，そのアドレスに制御を移す．

アドレスレジスタとデータレジスタは，使用される命令によって微妙に動作が異なります．たとえばmove命令でデータを移動するとき，転送先がアドレスレジスタの場合は，フラグの変化が発生しないのに，データレジスタの場合には内容によってフラグの値が変化します．

●アドレス空間

68000のアドレス空間は24ビット（16MB）です．68000はメモリマップドI/Oを採用しているので，I/Oはメモリ空間の一部に組み込まれます．

メモリマップドI/Oのとき，命令によってはリード・モディファイ・ライト動作を行うことに注意する必要があります．

68000では，0番地から3FFHまでを割り込みベクタテーブルとして使用します．リセット時には，このベクタテーブルの先頭からスタックポインタと実行開始アドレスを取り出します．このため，0～7番地はリセット時または電源投入時には値が固定（ROM）されていなければなりません．しかし，割り込みベクタはあとから動的に変更したい場合が多いでしょうから，RAMでないと困ります．この矛盾する要求は，通常はハードウェアで特殊な処理をして0～7番地だけをROMにマッピングし，残りはRAMになるようにして解決しているようです．この仕様はちょっと使いづらいですね．

CPUからは通常のアドレス信号のほかに，ファンクションコード（FC0,1,2）信号が3ビット出ていて，CPUがユーザーモードで動いているかスーパーバイザーモードで動いているか，プログラムの読み込みかデータの読み書きか，コプロセッサなどのCPUの機能拡張のための読み書きかを示しています．この信号は外付けのメモリ管理チップ用に出ているのですが，この信号をアドレス信号の一種と考えると，アドレス空間はもっと広がることになります．

68000のレジスタは32ビットですが，データバスは16ビットです．ですから，命令語で32ビットを直接操作しているときでも，CPUの外へ出る場合には16ビットに分割されて読み書きが行われます．

●アドレッシングモードと直交性

68000には次のようなアドレッシングモードがあります．

(1)データレジスタダイレクト
(2)アドレスレジスタダイレクト
(3)アドレスレジスタ間接
(4)ポストインクリメントつきアドレスレジスタ間接
(5)プレデクリメントつきアドレスレジスタ間接
(6)ディスプレイスメントつきアドレスレジスタ間接
(7)インデックスおよびディスプレイスメントつきアドレスレジスタ間接
(8)16ビット絶対アドレス
(9)32ビット絶対アドレス
(10)ディスプレイスメントつきプログラムカウンタ相対
(11)インデックスおよびディスプレイスメントつきプログラムカウンタ相対
(12)イミディエート
(13)クイックイミディエート
(14)インプリシットレジスタ

よく考えると，80x86にも対応するアドレッシングモードがあるのですが，68000はアドレス空間が単一であるということと，32ビットレジスタで直接表現できるということから，80x86よりも使いやすくなっています．

(1)データレジスタダイレクト
(2)アドレスレジスタダイレクト
(3)アドレスレジスタ間接

この3つは説明の必要がないでしょう．68000では，アドレスレジスタとデータレジスタがちょっと役割が異なっているので，別の扱いを受けています．データレジスタでは間接アドレッシングができません．

(4)ポストインクリメントつきアドレスレジスタ間接
(5)プレデクリメントつきアドレスレジスタ間接

この2つは，68000系がC言語に向いていることをよく表しているものです．これをC言語流に書くと，

X++：ポストインクリメントつきアドレスレジスタ間接
−−X：プレデクリメントつきアドレスレジスタ間接

となります．配列データの処理でよく使われる書き方です．これらに直接対応するアドレッシングモードを680x0

リスト2　データコピーの例

```
        68000                                   8086
    move.l #xxxx,a0                         lds     si,xxxx_ptr
    move.l #yyyy,a1                         les     di,yyyy_ptr
    move.w #cccc-#1,d0                      mov     cx,cccc
 loop1:
    move.b (a0)+,(a1)+                      rep movsb
    dbt    d0,loop1
```

系はもっているのです．つまり内容を操作したあとにアドレスレジスタを増やす，内容を操作する前にアドレスレジスタを減らすことができます．80x86系でもストリング命令がありますが，こちらのほうがより汎用になっています．

注意しなければならないのは，逆がないという点です．++XとX--に対応するモードはありませんので気をつけてください．また，メモリ上にあるポインタには，このような表現はできません．

あえてつけ加えておけば，アセンブラで書くときには+-は1つだけです．けっして(A0)++などと書いてはいけません．

かんたんなデータコピー（xxxxからyyyyへccccバイト）の例をあげておきます（リスト2）．

ここで8086の例もあげていますが，一般に80x86系と680x0系は反対のことが多く，アセンブラでもソースとデスティネーションの書き方が逆順になります．680x0系では左から右へ，80x86系では右から左へ動きます．

(6)ディスプレイスメントつきアドレスレジスタ間接
(7)インデックスおよびディスプレイスメントつきアドレスレジスタ間接
(8)16ビット絶対アドレス
(9)32ビット絶対アドレス

このあたりもわかりやすいですね．(6)はアドレスレジスタの中身で示されるアドレスから何バイト離れた場所かということですし，(7)はさらに別のレジスタの値を加えたアドレスを使用します．(8)と(9)は説明の必要もないでしょう．

(10)ディスプレイスメントつきプログラムカウンタ相対
(11)インデックスおよびディスプレイスメントつきプログラムカウンタ相対

プログラムカウンタ相対というのは，現在のプログラムカウンタの位置からどれだけ離れているかで位置を表す方法です．なぜこのような方法が必要かというと，アドレス空間のところでちょっと触れましたが，ファンクションコード信号を利用してメモリ管理が行われる場合には，プログラム空間での読み出しか，データ空間での読み書きかが区別されます．通常の方法ではアクセス時にはデータ空間として行われますが，(10)，(11)はプログラム空間として行われます．ですから管理の都合上プログラム空間に定数などを置く場合には，このような表現でプログラム空間を指定して読み出す必要があるのです．メモリ管理が行われない場合には，アドレスレジスタを使用するほうが便利です．

(12)イミディエート
(13)クイックイミディエート

この2つは定数を使用する場合です．クイックのほうは，値が1から8の場合に短い命令語ですますことができるようにしたものです．ループ処理などでのインデックス操作は，通常この範囲に入っていることが多いでしょう．

(14)インプリシットレジスタ

インプリシットとは，暗黙のうちに操作対象が決まっているものです．たとえば，LINK命令での操作対象がスタックに決まっているような場合です．

68010

68010は，68000に仮想記憶対応機能と仮想マシン対応機能を追加したCPUです．割り込みベクタテーブルの再配置が可能になり，「ROMかRAMか，それが問題だ」の0番地問題が解決しました．そのほかにもループ命令の改善などが行われています．

仮想記憶対応機能の追加とは，68000になかったページ不在やセグメント不在時へ対応する機能を追加し，再実行ができるようにしたことです．

次に仮想マシン対応機能の追加です．仮想マシンを実現するときには基本となるOSの上で別のOSをユーザーモードで動作させます．ユーザーモードで動いているOSがステータスレジスタ中のS（スーパーバイザー）ビットを参照する命令を発行します．68000ではこれがユーザーモードからも実行できてしまったため問題だったのですが，68010ではユーザーモードからは実行できなくし，そのかわりにCCR（コンディションコードレジスタ）へのアクセス命令を追加してあります．そのほかにも，ユーザーモードやスーパーバイザーモードの壁を超え

表7　68020で追加されたレジスタ

MSP	マスタスタックポインタ
VBR	ベクタベースレジスタ
SFC	ソースファンクションコードレジスタ
DFC	デスティネーションファンクションコードレジスタ
CACR	キャッシュ制御レジスタ
CAAR	キャッシュアドレスレジスタ

アセンブラ用語集
スタック　一時的にデータを格納する記憶領域を指し，サブルーチンや割り込み処理などの終了後に戻るアドレスやレジスタの内容を退避するのに使われる．

ての転送など，いくつかの命令が追加されました．

68020

アドレスバス，データバス，内部構成，すべてが32ビットになったCPUです．命令語は上位互換性があります．浮動小数点用のコプロセッサインターフェースを備え，周辺I/Oのバス幅に動的に対応して参照時のバス幅を変更できるダイナミックバスサイジング機構をもっています．

コプロセッサインターフェースが用意されたため，浮動小数点やメモリ管理のチップをかんたんに使用することができるようになりました．この68020からが，ほんとうの意味でワークステーションなどで使えるCPUということができるでしょう．

●レジスタ

MSPは，新たに追加された実行モード用のスタックポインタです．68020からはスーパーバイザーモードが割り込みモードとマスタモードと呼ばれる2つに分けられました．これまでの68000などのスーパーバイザーモードは割り込みモードと同じと考えられます．マスタモードの動作中に割り込みが発生した場合には割り込みモードへ移行します．このマスタモードをうまく利用すれば，割り込み処理ハンドラによるOS用のスタックの破壊を防ぐことができます．

VBRは，ベクタベースレジスタです．つまり，割り込みベクタの先頭アドレスをこのレジスタで定義することにより自由に配置することが可能となります．

SFCとDFCは，スーパーバイザーモード中からユーザーモードのアドレス空間へアクセスする必要が発生した場合などに使用されるモード指定レジスタです．

CACRとCAARは，キャッシュの制御用です．CACRはキャッシュをONにしたりフリーズしたり，クリアしたりするのに用いられます．リセット時にはキャッシュは無効な状態に設定されます．

68020には，256バイトの命令用内蔵キャッシュがあります．外部へアクセスした場合には3クロックかかるのが，キャッシュにヒットした場合，2クロックですみます．方式はダイレクトマッピング方式です．

●アドレス空間

68020のアドレス空間は32ビット（4GB）です．CPU自身にはメモリ管理機構はありませんが，外付けのメモリ管理チップ（68851）を使用すればメモリ管理できます．このときに注意が必要なのは，外でアドレス変換を行うのですから，CPUから出力されるアドレスは論理アドレスであるということです．出力されるアドレスが同じでも，状態しだいで別の実アドレスを示すことになります．

メモリ空間はユーザーには単一空間と考えられるので，シンプルなプログラムを作成することができます．

データバスも32ビットが基本となりました．ただし，周辺I/Oの要求に合わせて動的にバスサイズを調整できるので，16ビットI/Oや8ビットI/Oも混在することが可能です．

コプロセッサインターフェースは，実際にはファンクションコード信号で現在のアクセスがコプロセッサへ向けたものであることを指定し，バスを使用して行われます．ですから，バス上にはメモリやI/O要求以外の信号も出ます．

●メモリアラインメントの緩和

68020からデータのアラインメントが緩和されました．68000では奇数アドレスへのワードデータの参照がアドレスエラー例外を発生させていましたが，68020からは許されるようになりました．ただし，命令語は例外で命令のフェッチ（読み込み）が奇数番地からはじまることは許されていません．

これで，ほかのCPUとの間でデータをやり取りするときに，構造体の構成がアラインメント要求を満たしているかとか，通信のヘッダが奇数バイトではないかと思い悩まなくてもよくなりました．

たしかにハード的には，アラインメントが一致している場合のほうが速いので，合わせたほうがよいことはまちがいないのですが，動かないよりはちゃんと動いてくれるほうがありがたいのでよいことだと思います．

ハード的に速いというのは，次のようなことです．たとえば，16ビットサイズのメモリへ16ビットの要求を出したとします．メモリは自分のサイズ分のデータを読み書きすることしかできません．アラインメントがあっている場合には，要求は1回のアクセスですみますが，合っていない場合には最初のワードから上位バイト，次のワードから下位バイトと，2回に分けて読み書きする必要があるのです．このように，メモリ要求の回数が増えることは速度の低下につながりますから，アラインメントが合っているほうが速いわけです．

とはいっても，プログラム側でバイト単位で取り出して再構成して処理す

るよりは，ミスアラインメントであってもCPUが直接読み書きするほうが速いことはまちがいありません．

まあ，ある程度経験を積んだ人が設計する場合には，受け渡しデータのアラインメントを合わせることはあたりまえでしょう．

●アドレッシングモードの増加

68020では，アドレッシングモードがさらに強化されました．

メモリ間接のプリ／ポスト・インデックス
プログラムカウンタメモリ間接プリ／ポスト・インデックス

上記の追加とディスプレイスメント部分の32ビット化，さらにインデックスに定数値をかけることのできるスケールインデックスが追加されました．

プリ／ポスト・インデックスとは，間接値の内容を取り出すアドレスにインデックスを加えるか，取り出した値にインデックスを加えるかを指定できる間接参照です．つまりポインタの先にある配列にアクセスするのか，ポインタの配列中のポインタで示される場所にアクセスするのかの違いです．

スケールインデックスのサポートで配列の操作が楽になります．使用可能なスケール値は，1，2，4，8の4種類です．さらにアドレスレジスタが指定されない（アドレスレジスタ部分は0とみなされる）インデックスモードが許されるのでデータレジスタで間接アクセスができます．

ここまでくると，アセンブラを書いていても気分は高級言語感覚になってしまうのは私だけでしょうか？　32ビットCISCチップのアセンブラなんて楽なもんだ．

表8　アセンブラから見た68020のアドレッシングモード

<table>
<tr><td>1.</td><td>#data</td><td>定数値</td></tr>
<tr><td>2.</td><td>Rn</td><td>レジスタ直接</td></tr>
<tr><td>3.</td><td>(An)+</td><td>ポストインクリメント</td></tr>
<tr><td>4.</td><td>−(An)</td><td>プリデクリメント</td></tr>
<tr><td colspan="3">プリインデックス系統</td></tr>
<tr><td>5.</td><td>([disp,]An[,Rn['*'scale]])</td><td>アドレスレジスタ間接関係</td></tr>
<tr><td>6.</td><td>([disp,]Rn['*'scale])</td><td>レジスタ間接関係</td></tr>
<tr><td>7.</td><td>(disp,'PC'[,Rn['*'scale]])</td><td>PC 間接関係</td></tr>
<tr><td>8.</td><td>('['mode1']',disp)</td><td>mode1は5．6．7のどれか</td></tr>
<tr><td colspan="3">ポストインデックス系統</td></tr>
<tr><td>9.</td><td colspan="2">('['[disp,] {An | 'PC' | 'ZPC'} ']'[[,disp],Rn['*'scale]])</td></tr>
</table>

9.の例：ベクタテーブルのDO番目の項目にジャンプする
　　jmp　([vect).1,do.w *4])

ここで，アセンブラから見た68020のアドレッシングモードを思いっきりまとめてしまうと表8のようになります．なお，Anはアドレスレジスタ，Rnはアドレスレジスタまたはデータレジスタ，dispはアドレスまたは変位，scaleは1，2，4，8の中の1つです．[]部分は省略可能，{ | }部分はどれかを選択，''で囲まれた部分はそのまま指定します．

68030

ここからは私自身がさわったことのないCPUですので，マニュアルや本の上で調べたことのまとめです．

68030は68020にメモリ管理とデータキャッシュを追加したCPUです．

命令の実行は3段のパイプライン化され，CPU内部では命令用のバスとオペランド用のバスを独立にもったハーバードアーキテクチャと呼ばれる手法を用いています．つまり，バスが完全に2組用意されているということで，CPU内部では命令とデータを完全に分離して独立に処理することができます．

キャッシュはそれぞれのバスに256バイトずつ独立に用意され，データキャッシュのほうはライトスルー方式になっています．

●メモリ管理

68030に内蔵されたメモリ管理機能は，外付けのメモリ管理コプロセッサ68851のサブセットです．内蔵された機能は，論理アドレスから物理アドレスへの変換機能だけでアクセスレベル機能やブレークポイント機能はサポートされていません．

68851にはなかった透過変換という機能が追加されています．これはメモリ管理のオーバーヘッドを避けるために，論理アドレスをそのまま物理アドレスとして使用するブロックを指定できるようにしたものです．この機能をOS部分のプログラムや画像処理用のバッファに使用すれば速度の改善が期待できます．

メモリ管理方法は，ページ単位の管理とそれを行わない場所を指定する透過変換ブロック指定が可能です．

68040

68040は，発表から実際に使えるようになるまでに時間がかかりすぎたようです．その間にRISCチップを使用

アセンブラ用語集
ストリング　メモリブロック同様に連続するメモリ上のデータ列を指す．

したワークステーションが話題をさらい，ワークステーションのRISC化が決定づけられたように思えます．

68040は，浮動小数点ユニット，メモリ管理ユニット，4KBのキャッシュ2組を組み込んだ32ビットCPUです．

浮動小数点ユニットの内蔵と整数ユニットの高速化で，計算能力が大幅に向上しているようです．浮動小数点ユニットは，68882とほぼ同じ機能をもっています．しかし，コプロセッサインターフェースが不要になったのと，内部に64×8ビットの乗算器を内蔵していることでより高速化しています．

メモリ管理ユニットは，命令用とデータ用の2系統に分かれています．それぞれにページ単位の管理と透過変換機能をサポートしています．

キャッシュは4KBに増やされ，さらに命令用とデータ用の2系統が用意されています．基本のアルゴリズムがコピーバックに変更され，キャッシュにヒットしているかぎりメモリへの書込みを行わなくなります．ミスヒットで書き戻される場合か，フラッシュ命令で書き戻されるまで，メモリ上のデータは更新されません．設定しだいではライトスルー方式も選べます．

68040には，複数のバスマスタをサポートするためのバススヌーピング機構が内蔵されています．この機能により，DMAコントローラなどを使用してもキャッシュの中身との整合性が保たれるようにすることができます．

RISC系 ◆ アセンブラは使わない？

ワークステーションで花盛りのRISC CPUの話も少しだけのぞいてみましょう．

RISC CPUは，CPUの構成を単純化し，動作周波数を高くして高速の演算ができるようになっています．ひとつひとつの命令は単純で，680x0系の間接インデックスなんてことはまったくできません．

RISCが市民権を得ることができたのは，最近のコンパイラ技術の進歩によるところが非常に大きいといえます．RISCチップの命令は，まるでアセンブラを作るためのアセンブラというようなレベルです．

しかし考え方を変えると，覚える命令語数が少なくていいのだからアセンブラとしてはわかりやすいのかもしれません．コンパイラと競争できるような最適化コードが書けるかどうかは別として，やってみるとおもしろいかもしれません．

サブルーチンコールなども直接の命令がないのが普通で，レジスタにリターンアドレスを入れてジャンプしたりします．割り込みなどもすべて同じ場所へ飛び，ソフトで要因解析して対応する処理へ分岐します．

ただし，RISCといっても現在確定した基準があるわけではないので，各CPUメーカーがRISCだといえばRISCに分類されるというのが現状のようです．

インテルやモトローラもCISC CPUだけではなくRISC CPUも出しています．

SPARC

ご存じSunのSPARCです．64種の基本命令をもっています．

SPARCは，Sun Microsystems, Inc.だけでなく，Bipolar Integrated Technology, Cypress Semiconductor, Fujitsu Microelectronics, LSI Logicなどの米国メーカーが互換CPUを作っています．

ただしSPARCの場合，互換CPUであってもチップ互換ではありません．チップ構成や機能などもそれぞれ異なっているようです．SPARCで決めているのは，命令セット，アドレッシングモード，MMUなどのソフトウェアの互換性を取るために必要な仕様だけです．

SPARCの特徴として最初にあげられるのは，レジスタウィンドウでしょう．SPARCは，32ビットレジスタを120個または136個も内蔵しています．このうちの8個はグローバルなレジスタ（r0〜r7）で，残りは手続きごとに割り当てて使用するレジスタです．

1つのレジスタウインドウは24個構成で，8個が呼び出し元とのオーバーラップレジスタ（r24〜r31）で，8個がローカルレジスタ（r16〜r23），あとの8個がその手続きが呼び出す手続きとのオーバーラップ用（r8〜r15）になります．この機構を使うことで，パラメータをスタックを使うことなくレジスタ渡しできるようになります．

SPARCにはcall命令が用意されています．call命令は戻り番地をr15に書き込みます．呼ばれた側では，save命令でレジスタウィンドウを切

アセンブラ用語集
セグメント　8086のアドレス空間は1MBある．このアドレスを指定する方法としてSegmentという概念を導入した．

り替えます．レジスタウィンドウの戻しはrestore命令で行います．

しかし，このレジスタウィンドウはほんとうに手続き単位で使われているのでしょうかね？

なお，jmplというレジスタに現在アドレスを設定後ジャンプする命令もあります．

SPARCの仕様で決まっているメモリ管理は，3レベルページングを用いるReference MMUという方法です．しかし，SunOSで使用しているのはSun-3のころからのMMUに対応したものとなっているため，Reference MMUを搭載するメーカーはSunOSの書き換えが必要です．

どのような命令があるかをかんたんに見てみると，加算，減算，乗算，論理演算，シフト，ロード，ストア，分岐，トラップ，特殊レジスタの操作，コプロセッサ命令などです．除算は見あたりません．ロード／ストア関係が，サイズや浮動小数点レジスタごとにたくさん用意されています．

割り込みは，ベクタテーブル(255個)を通して要因でジャンプできるようになっています．

SPARCは，どうもパソコンでのMS-DOS+80x86のように，ワークステーションでのUNIX+SPARCという標準を作りそうな気がします．SPARC用のチップセットなども開発され，IBM PCのクローンマシンのように，かんたんにワークステーションができる時代が以外と早くやってくるのではないでしょうか．それを期待させるだけのオープンな戦略がSPARCにはあるように思えます．

R3000

こちらのほうはソニーのNEWSで使用されているCPUです．メーカーは，MIPS Computer Systemsです．製造権をもっているのは，Integrated Device Technology，LSI Logic，Performance Semiconductor，Siemens，日本電気，ソニーの6社です．

おもしろいことに，R3000も出所はアメリカのスタンフォード大学です．Sunもここの大学から出てきた会社ですから，よほどすばらしい大学なのでしょうね．

R3000は，互換CPU間で端子レベルの互換性があります．このへんは方向性の差を感じさせますね．

R3000は32ビットレジスタを32本もっていて，74個の命令（ロード，ストア，加減乗除，論理演算，シフト，分岐命令，コプロセッサ命令）をもっています．除算ももっています．

MMUと，外付けのキャッシュ用の制御回路も内蔵しています．MMUは64エントリのアドレス変換バッファをもち，4GBの仮想空間をサポートしています．

サブルーチンコールはJump And Link命令で行い，リターンアドレスはr31に入ります．割り込みは1か所に飛んでくるようです．そこで要因解析をして必要な処理に分岐します．

節操がないというかがんばっているというか，R3000はリセット時にビッグエンディアンにするかリトルエンディアンにするかを設定可能です．680x0系にも80x86系にも合わせることができるのですね，このCPUは．

■最後に

駆け足で80x86系と680x0系のCPUを眺めてみました．だいたいの感じがつかめていただけたでしょうか？

集積度の向上でCPU作りも幅が広くなり，速度や管理機能の向上だけではなく，周辺LSIをチップ内に取り込むことも盛んに行われています．

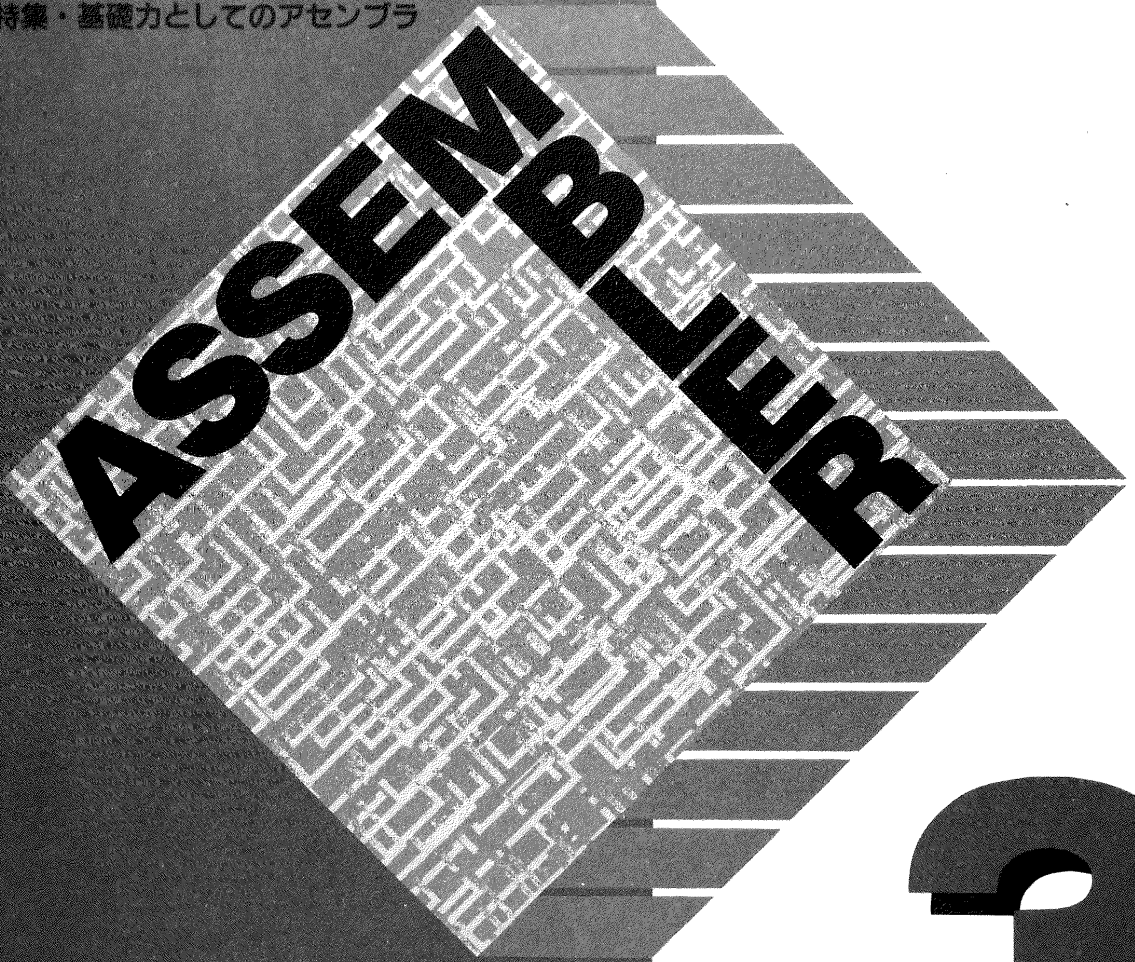
そのうち，RISC系のCPUも組込み用の分野に進出してくるでしょう．

多様化の波の中で，すべてのCPUを知ることは不可能になっていくでしょうが，1つのCPUに精通することができれば，意外とほかのCPUを勉強することはかんたんなことのように思います．

まずは，あなたの目の前にあるCPUにチャレンジしてみることをお勧めします．

わからないことが出てきたら，日経MIXのアセンブラ会議へおいでください．たぶん問題解決に協力できると思います．:)

アセンブラ用語集
セグメントアドレス　アドレスを指定したとき，この値を4ビット左にシフトしてオフセットアドレスを加えて実アドレスを決める．



3

高級言語とアセンブラのプログラミング

リンク&インラインアセンブラ

加野島英渡

現在，もっともポピュラーなアセンブリ言語の利用法が，C言語などの高級言語で書かれたプログラムの一部をアセンブリ言語で記述するというものである．このようなミクスドランゲージプログラミングにおけるリンクの詳細と，インラインアセンブラ技法について解説する．また，OBJファイルを解析するツールとして，オブジェクトモジュールダンププログラムを紹介する．

リンクとはなにか

未定義シンボル参照を解決し，実行モジュールを生成

高級言語とのリンクとインラインアセンブラについて書けとの仰せではあるが，敵を倒すにはまず敵を知らなければならぬ．リンクとはそもいかなるものか，リンカはいったいなにをしておるのか，ちょっとおさらいしてみようじゃないの．

リンク：

リロケータブルモジュールやライブラリを結合して未定義シンボル参照を解決し，モジュール内相対番地から絶対番地への変換を行い，実行可能なモジュールを生成すること．リンカとはこの操作を行うためのプログラム．

…なんだそうだ．MASMやCなどの言語処理系の出力するOBJという拡張子のついたファイルは，このリロケータブルモジュールという形になっている．

なぜ，直接実行可能なオブジェクトを出さないのか？　ソースファイルが1本であれば話は簡単なのだが，大規模なソフトウェアや，それほど大きくなくても複数の人間が分担して開発するソフトウェアでは，プログラムをいくつかのソースファイルにわけて開発することが多い．このような形態で開発されているソフトウェアでは，現在コンパイル中のソースファイル内には定義されていないシンボル（Cでいう関数や変数，アセンブラでは飛び先のラベルなど）が呼び出され，あるいは参照されることがある．このシンボルの値（アドレス）は当然ながらコンパイル中にはわからない．

また，このようにいくつかのソースファイルにわけて開発しているものでは，言語処理系はシンボルのコンパイル単位での相対的な位置こそわかるものの，どのような順序で1つにまとめられるかがわからないので，他のすべてのモジュールを通してみた場合の絶対的な位置はわからない．いやな例だが，クラスの成績順はわかっていても，ほかのクラスの全データがなければ，学年をとおしての順位はわからないというのと同じである．

これでは実行可能なオブジェクトなど作りようがないから，そのかわりにあとで全部の情報がそろったときにつじつまあわせができるような情報を出力しておく．この状態のものをリロケータブルモジュールと呼んでいるわけである．このように，言語処理系の出力するオブジェクトはなんとも心許ない状態でこの世に生を受けてくる．もちろん，こんなものをメモリに落としたって動くはずがない．メモリはそれほど賢くない．

さて，そこでリンカの登場とあいなる．リンカはこういった状態のリロケータブルモジュールをすべて読み込み，

図1　リロケータブルモジュールの作成

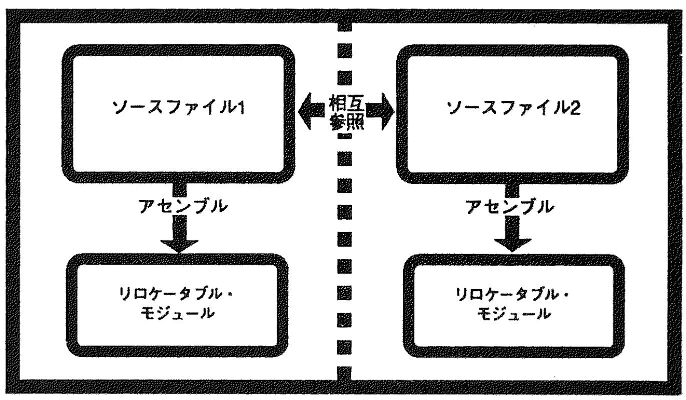
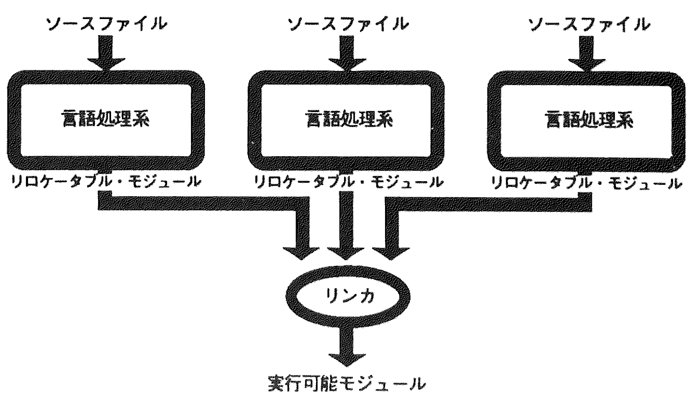


図2　オブジェクトモジュールのリンク



アセンブラ用語集
セグメントレジスタ　コード領域　スタック領域やデータ領域のセグメントアドレスを格納するレジスタ．

リスト1　a.asm

```
data    segment word public 'data'
data    ends

stack   segment stack
stack   ends

DGROUP  group   data, stack

extrn   b_proc:near          ; (1)

code    segment byte public 'code'
    assume  cs:code

a_proc:
    call    b_proc
    ret
code    ends

    end
```

リスト2　b.asm

```
data    segment word public 'data'
data    ends

stack   segment stack
stack   ends

DGROUP  group   data, stack

public  b_proc               ; (2)

code    segment byte public 'code'
    assume  cs:code

b_proc:
    dec     ax
    ret
code    ends

    end
```

モジュール間の参照関係を解決し，モジュール内相対であったシンボルアドレスを絶対アドレスに変換して，MS-DOSの場合はさらに先頭にEXEヘッダなる怪しの護符を貼りつけて実行可能なモジュールを作ってくれる．

ちょっと，リスト1,2の2つのソースを見てもらいたい．

説明のためなので，プログラム自体にはなんの意味もない．

a.asmのa_procからはb.asmにあるb_procを呼び出している．しかし，b.asmは別のソースファイルなので，a.asmのほうで「ここにはないけどほかのファイルにnearで呼ばれる形のb_procというのがあるよ」という宣言をしている（1）．逆にb.asmの中では，b_procは外部から呼び出されるという宣言をしている（2）．

ここで問題．a_procとb_procのオフセットアドレスを答えよ．

この段階では「わからない」というのが正解．それぞれのモジュールの先頭からの相対アドレスはわかっているが，リンク時にどちらが先にリンクされるかで実行モジュール内のアドレスが変わってくる．図3,4は，それぞれのリンク順序での実行モジュールのダンプリストの一部である．callのオペランドフィールドがちゃんと解決されていることも確認できる．「a_proc先頭」（E8H）の次の2バイトがオペランドにあたる．

図3　a.objが先

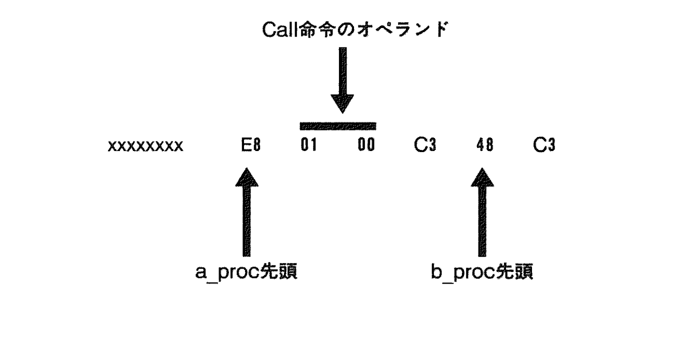
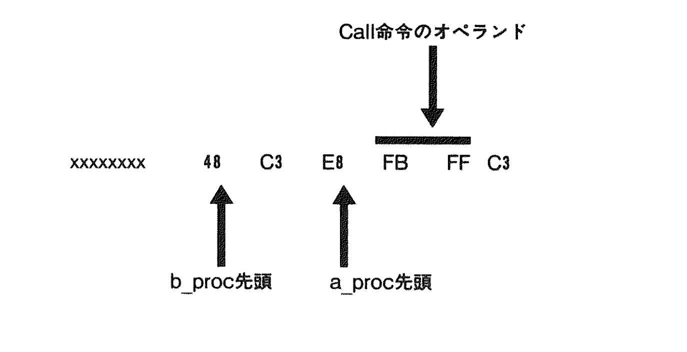


図4　b.objが先



では，リロケータブルモジュールを少しくわしく見ていこう．いちばんなじみの深いマイクロソフトのものを例にとる．マイクロソフトでは，このリロケータブルモジュールを「オブジェクトモジュール」と呼んでいるようなので，こちらもそれにならうとしよう．

OBJファイルの構造

◆可変長のレコードを単位として構成

あとでお目にかけるが，オブジェクトモジュールを少し見やすくダンプするツールをこの原稿を書くために作った．図5は，そのオブジェクトモジュールダンププログラムのオブジェクトモジュール（え〜い，ややこしいわいっ）の先頭部分である．「ふむふむなるほど」とすらすら読めるという人は，すごいけど少しビョーキ．

オブジェクトモジュールは，可変長のレコードと呼ばれる単位で構成されている．レコードは全部で14種類あり，それぞれのレコードが別々の情報を管理している．1つのオブジェクトモジュールの中に何回も出てくるレコードもあれば，たった1回決まった位置にしかでてこないレコードもある．

1つのレコードは図6のような形態になっており，これがいくつか集まってオブジェクトモジュールを構成している．先頭は必ずTHEADRレコードであり，MODENDレコードで終わる（これらのレコードの詳細は後述）．

オブジェクトモジュールは，このようなレコードからのみ構成されている．別にヘッダのようなものはなく，いきなりTHEADRレコードからデータがはじまっている．

図5　オブジェクトモジュールの先頭部分

```
00000000 80 09 00 07 4F 42 4A 44 55 4D 50 5F 96 23 00 00
00000010 06 43 47 52 4F 55 50 04 54 45 58 54 06 44 47 52
00000020 4F 55 50 04 44 41 54 41 03 42 53 53 04 43 4F 44
00000030 45 29 98 07 00 28 5B 01 03 07 01 D2 98 07 00 48
00000040 3C 01 05 05 01 D1 98 07 00 48 02 00 06 05 01 0B
00000050 9A 04 00 02 FF 01 60 9A 06 00 04 FF 02 FF 03 59
```

図6　レコード

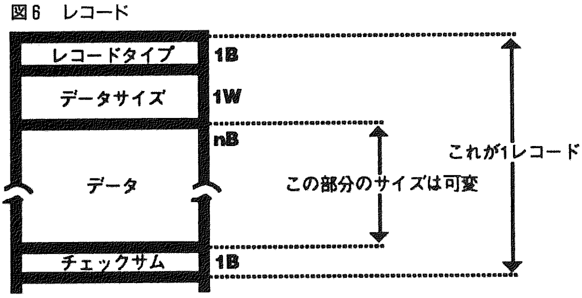


図7　オブジェクトモジュールの構成

ID	NAME	概略説明
80	THEADR	処理系ヘッダレコード．オブジェクトモジュール名が入っている．
88	COMENT	コメントレコード．コメントなどの文字列とその属性など．
8A	MODEND	モジュール終了レコード．オブジェクトモジュールの最後に置かれる．
8C	EXTDEF	外部名定義レコード．外部参照シンボルの名前などが入っている．
8E	TYPDEF	タイプ定義レコード．PUBDEF,EXTDEFなどの詳細情報が入っている．
90	PUBDEF	パブリック名定義レコード．パブリックシンボル名や相対アドレス．
94	LINNUM	行番号レコード．ソースファイル中の行番号などのデータ．
96	LNAMES	シンボル一覧レコード．SEGDEF,GRPDEFから参照される名前の一覧．
98	SEGDEF	セグメント定義レコード．論理セグメント名，サイズ，属性など．
9A	GRPDEF	グループ定義レコード．セグメントグループに関するデータ．
9C	FIXUPP	フィックスアップレコード．シンボル参照の解決のためのデータ．
A0	LEDATA	論理列挙レコード．データや実行可能なコードが入っている．
A2	LIDATA	論理反復レコード．上に同じ，ただし反復されるもの．
B0	COMDEF	コモン名定義レコード．マイクロソフト拡張レコードフォーマット．

各レコードの構造

これでわかるように，オブジェクトモジュールの形になってしまったものは，それがどの言語で記述されたものかはわからなくなっている．オブジェクトモジュールが保持している情報は，「この論理セグメントの名前はこれこれ，同じ名前のものはこのように結合してください」「このデータは以下のパターンが反復されています」「アドレスが確定したらここをこのような規則で書き換えてください」といったものである．なにで書かれたかなどという情報はリンク時には必要ないのだ．使用された言語の名前はコメントの形で入っているにすぎない．

アセンブラで書かれていようが，Cで書かれていようが，FORTRANだろうが，事情は同じ．オブジェクトモジュールが，このような性格になっていることからミクスドランゲージプログラミング（1本のプログラムを複数の言語で記述すること）が可能になってくるのである．言語にはそれぞれ得

アセンブラ用語集
ソース　ディスティネーションに対して処理のもととなるオペランドを指す．

図8 各レコードの構造



手不得手があるから，それぞれの部分にもっとも適した言語を使ってプログラムを開発できるというのはありがたい．

ところで，身近なところではCの一部の関数をアセンブラで記述するのもミクスドランゲージプログラミングの一種である．C+アセンブラという，最近ではわりとポピュラーな手法について，ソースの書き方，リンクの方法などについて検討していこう．

C言語とのリンク(Turbo C, MS-C)

レジスタ保存と引数の受け取り方が問題

Cは非常に効率のいいコードを出力する．とくに最近のCコンパイラは最適化が強力になっているので，人に聞いた話しだが，新人のプログラマよりよほどいいコードを出すこともあるという．いやな時代だなぁ．

だからといってアセンブラの必要性がまったくなくなったかというと，そうではない．いくら最新のCコンパイラといっても，CPUフラグや最適な命令を上手に使ってちゃんと書かれたアセンブラプログラムの効率にはまだまだおよばないし，ビット操作などの非常に低レベルな処理はやはりアセンブラのほうが得意だ．

ただ，速度に影響を与える部分は全体の10パーセント程度にすぎないといわれているので，開発効率を考えると，できるだけ多くの部分をCで書き，スピードを要求される部分のみ根性を入れてアセンブラで書くというのが上手な使い方だろう．

さて，すべてCで記述している場合はとくに考えなくてもよかった問題が，

アセンブラ用語集
ソフトウェア割り込み　ハードウエア割り込み同様に，この割り込みが発生すると処理ルーチンに処理を移す．

アセンブラで関数を書こうとするといろいろと出てくる．レジスタ保存や引数の受け取り方などだ．では，Cの関数をアセンブラで書く場合の技術的な情報について考えてみよう．そのためには，まずCの関数がどのように動いているかを知る必要がある．

引数の渡され方

Cでは，関数への引数はスタックを経由して渡される．LSIC-86や_first-call宣言をつけたMS-C 6.0の関数のようにレジスタで渡すものもあるが，スタック経由が一般的だ．以下，Turbo CとMS-Cを対象にして話を進めていこう．

引数はスタックにいちばん右側のものから順にPushされ，目的の関数が呼び出される．このときのスタックのようすを図9に書いてみる．話を簡単にするため，引数はすべてintとして考える．スタックポインタSPは，データがPushされるたびに高位番地から低位番地へ移動する．

おー，簡単簡単，Popすればデータが取れるじゃないの……おっとそれはだめ．Cでは引数はそれをPushしたほうで捨てることになっている．決して呼び出されたほうでデータを捨ててしまってはいけないし，呼び出した関数に戻るときに“ret <n>”を使ってはいけない．

戻り値の返し方

引数と違って，戻り値はレジスタで返される．使用されるレジスタはAXとDXで，16ビット以下はAXが使われ，32ビットまでの値は上位ワードがDXに，下位ワードがAXにセットされる．これ以上の値や構造体などは，そのデータがおかれているアドレスをこれらのレジスタで返す．この場合，データモデルがスモールならばオフセットのみをAXで返し，ラージならばDX:AXで返す．

以上のように，Cの関数ではスタック上に置かれた引数を参照し，レジスタに値を返す．アセンブラ記述の関数でも，これとまったく同じようにすればいいのだ．

レジスタの保存

Cの関数をアセンブラで書く場合に，もうひとつ考えておかなくてはならないことがある．それはレジスタの保存だ．CSは当然として，SS，DSの各セグメントレジスタと，BP，SI，DIは保存しなくてはならない(注1)．「保存しなくてはならない」というのは，必ずPushしなくてはならないという意味ではない．内部でこれらのレジスタを使わないのであれば，わざわざPushする必要はない．ただ，このうちBPだけは引数を参照するのに使用するので，たいていの場合は保存することになるだろう．また，フラグのうちディレクションフラグは保存しておく必要がある．フラグ1ビットだけのPushはできないので，フラグレジスタ全体を保存することになる．

引数の参照

先に述べたように引数はスタック経由で渡されるため，BPを使ったレジスタ間接アドレシングでアクセスするのが一般的である．図11を見てもらいたい．すでにBPをPushしているため，最初の引数は“［BP+4］”で参照することになる．同様に二番目の引数は“［BP+6］”，三番目の引数は

図9　スタックのようす

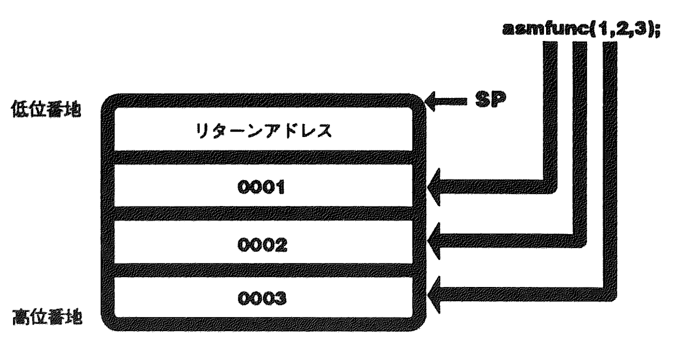
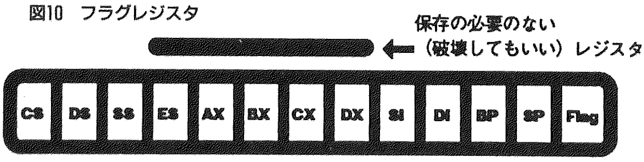


図10　フラグレジスタ



注1：これはTurbo CとMS-Cの話，他のCコンパイラでは事情の違うものもある．

アセンブラ用語集
ダブルワード　32ビット（4バイト）で表せる値をワードといい，単精度浮動小数点などを表すのに使われる．

図11　スタックの状態

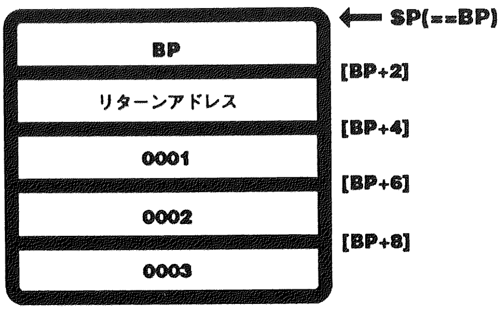
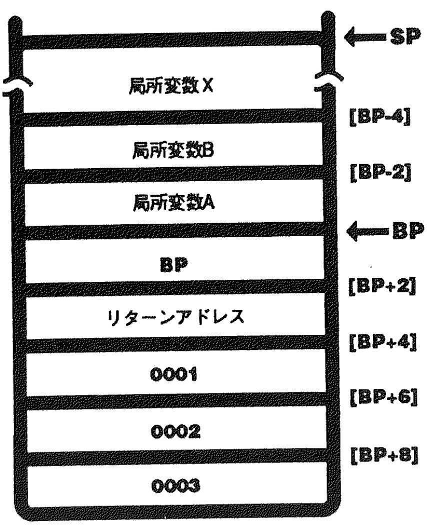


図12　関数の入口と出口

```
push    bp
mov     bp, sp
(sub    sp, n)
.....
(mov    sp, bp)
pop     bp
ret
```

図13　関数内での局所変数の確保



“［BP+8］”で参照する．

つまり，関数呼び出しのときにスタックに詰まれる戻りアドレスのサイズ（スモールモデルなら2，ラージモデルなら4）に2（BPのPush分）を加えたものが，最初の引数を参照する（BPからの）オフセットである．図11は，関数入口のスモールモデルでのスタックの状態である．

関数の入り口と出口

では以上の知識から関数の入り口と出口を書いてみよう．

関数内で局所変数を確保する場合は，括弧のような処理が必要になる．SPをかさ上げしてそこに局所変数領域を作り，BPからの負のオフセットでアクセスする．もちろん出口ではSPの値を戻しておかなければならない．

実際のアセンブラソース

では，ソースファイルを作ってみよう．といってもちゃんとしたソースファイルを作ろうとすると，セグメント名やグループなどめんどうなことがいろいろとある．そこで究極のてぬきうどん（あ，うどんがきらいな人はここに「そば」とパッチをあてて読むコト），Cコンパイラにソースを作ってもらうことにしよう．MS-CにはCのソースからアセンブラソースを出力する/Faというオプションがある．このオプションをつけて空のC関数を1つ与えると，自動的に完全なアセンブラソースのひな型ができあがるという寸法だ（リスト3）．Turbo Cにも，-Sというアセンブラソースを出すオプションがあるのだが，出てきたソースがいかにも面妖で尻ごみしてしまった．MS-Cで出力したソースをベースにして書いたアセンブラプログラムのオブジェクトモジュールは，当然ながらTurbo Cの出力したオブジェクトモジュールともリンクすることができる．

```
A>cl /Fa /c asmfunc.c
```

ほんとうはもう少しいろいろと情報が出てくるのだが，不要と思われる部分は削除してある（リスト4）．この（1）の部分に必要なプログラムを書き込めばいいのだ．

アセンブラ用語集
ディスティネーション　ソースに対して処理の対象となるオペランド．

リスト3 asmfunc.c

```
asmfunc()
{
}
```

リスト4 asmfunc.asm

```
_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP    _BSS, _DATA
    ASSUME DS: DGROUP, SS: DGROUP

_TEXT       SEGMENT
    ASSUME  CS: _TEXT

    PUBLIC  _asmfunc

_asmfunc    PROC NEAR
                ...................... (1)
_asmfunc    ENDP

_TEXT   ENDS

        END
```

リスト5 BCASM.ASM

```
_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP    _BSS, _DATA
    ASSUME DS: DGROUP, SS: DGROUP

_TEXT       SEGMENT
    ASSUME  CS: _TEXT

PUBLIC      _bit_count

_bit_count      PROC NEAR
    push    bp
    mov bp, sp
;
    xor ax, ax
    mov bx, [bp+4]
    mov cx, 16
_lp:
    shl bx, 1
    adc ax, 0
    loop    _lp
;
    pop bp
    ret

_bit_count  ENDP

_TEXT   ENDS

        END
```

リスト6 BC.C

```
#include    <stdlib.h>

int     bit_count(int);

void    main(int argc, char **argv)
{
int     result;
    if(argc != 2)
    {
        printf("Usage: A>BC <number>¥n");
        exit(1);
    }
    result = atoi(argv[1]);
    printf("Answer = %d¥n", bit_count(result));
    exit(0);
}
```

では，実際にやってみよう．intで与えられた引数のうちの‘1’のビットの数を返す関数“bit_count”をアセンブラで記述する（リスト5）．呼び出すほうはCで書き（リスト6），コマンドライン引数から数値を1つ取り，それを引数にして“bit_count”を呼び出し，結果を表示する．

“bit_count”のアルゴリズム（というほどのものではない）は，1ビットずつ左にシフトして最上位ビットをキャリーフラグに入れ，結果を入れるレジスタにキャリーとともに0を加えることでカウントするというごくあたりまえのものだ．

コンパイルとリンクの手順

■ Turbo C (Ver.2.0)

```
A>tcc  bc.c  bcasm.asm
```

■ MS-C (Ver.6.0)

```
A>cl /c bc.c bcasm.asm
A>link bc+bcasm;
```

※MS-Cでわざわざリンクを別にしているのは，clがリンカを呼び出すときに/NOI[GNORECASE]オプションをつけるからだ．キャンセルする方法がないかといろいろやってみたのだが，わからなかった．マニュアルの「ミクスドランゲージプログラミング」の

アセンブラ用語集
データセグメント　プログラムで使われるデータが格納されるアドレスのセグメント．

リスト7　BCFUNC.C

```
int     bit_count(unsigned int n)
{
int     count, i;

    for(count = i = 0 ; i < 16 ; i++)
    {
        if(n & 0x8000)  count++;
        n = n << 1;
    }
    return  count;
}
```

リスト8　BCMSC.ASM(MS-C)

```
_bit_count  PROC NEAR
    push    bp
    mov     bp, sp
    sub     sp, 4
    sub     ax, ax
    mov     WORD PTR [bp-2], ax  ;i
    mov     WORD PTR [bp-4], ax  ;count
    jmp     SHORT $F277
$FC278:
    test    BYTE PTR [bp+5], 128
    je      $I280
    inc     WORD PTR [bp-4] ;count
$I280:
    shl     WORD PTR [bp+4], 1   ;n
    inc     WORD PTR [bp-2] ;i
$F277:
    cmp     WORD PTR [bp-2], 16  ;i
    jl      $FC278
    mov     ax, WORD PTR [bp-4]  ;count
    mov     sp, bp
    pop     bp
    ret
```

リスト9　BCTC.ASM(Turbo C)

```
_bit_count  proc    near
    push    bp
    mov     bp, sp
    push    si
    push    di
    xor     di, di
    mov     si, di
    jmp     short @5
@4:
    test    word ptr [bp+4], -32768
    je      @6
    inc     si
@6:
    mov     ax, word ptr [bp+4]
    shl     ax, 1
    mov     word ptr [bp+4], ax
@3:
    inc     di
@5:
    cmp     di, 16
    jl      @4
@2:
    mov     ax, si
    jmp     short @1
@1:
    pop     di
    pop     si
    pop     bp
    ret
```

リスト10　BCLSIC.ASM(LSIC-86/参考)

```
bit_count_::
    PUSH    CX
    XOR     BX, BX
    MOV     CX, BX
    JMP     _1
_2:
    TEST    AX, -32768
    JE      _7
    INC     CX
_7:
    SHL     AX, 1
    INC     BX
_1:
    CMP     BX, 16
    JL      _2
    MOV     AX, CX
    POP     CX
    RET
```

項を見ると，これが困る場合はリンクを別にせよと書いてあるので，キャンセルする方法はないのかもしれない．

実行結果

```
A>bc 1
Answer = 1

A>bc 3
Answer = 2

A>bc 255
Answer = 8
```

当然ながら，どちらでコンパイルしたものも同じ結果になった．めでたしめでたし．

ところで，試しにこの関数をCで書いてみよう（リスト7）．ビットカウントにはもう少しいい方法があるのだが，ここではアセンブラ記述の関数と同じやり方をとっている．

それぞれのコンパイラで生成されたコードはリスト8,9,10のようになっ

た．特別な最適化オプションなどはつけていないが，MS-Cにはスタックチェックを省略するコンパイルオプション/Gsを指定した．ことわっておくが，コンパイラの性能比較ではなく，アセンブラの効率のよさを見るためであるから誤解のないように．

Turbo Cでは，黙っているとint型の変数をSI, DIに割り付けるようだ．MS-Cでは明確にregisterと宣言しなくてはならないのだな．ふぅむ，MS-Cではnの最上位ビットを見るのにバイト値を使っているのか（コンパイラの性能比較でないっちゅうのに）．

インラインアセンブラ

コンパイラによって記述方法はいろいろ

本来は，このようにアセンブラ記述部分は別ファイルにして開発するものなのだが，ほんの数バイトのモジュールをわざわざ別ファイルにするのもちょっとめんどうだし，ファイルが増えればそれだけ管理も難しくなる．また，場合によっては速度を稼ぐためにCの関数の中に直接アセンブラ記述をもぐりこませたいことがある．関数にしてしまうと呼び出しと復帰のためのcall/returnや，引数を取り出すためのレジスタ操作などのオーバーヘッドが生じ，小さな処理だとなんのためにアセンブラで書いたのかわからなくなってしまうこともあるからだ．

Turbo C，MS-Cとも，このような要求を満たすように「インラインアセンブル機能」をもっている．この機能は，Cのソース中に直接埋め込まれたアセンブラのソースを，コンパイル時に自動的にアセンブルしオブジェクトモジュールを生成してくれるというものだ．Cのソースの中なので，前項のようなセグメント定義などは不要だから，手軽にアセンブラソースを書き込むことができる．しかもCのソースの中に直接書けるため，アセンブラ記述のソースファイルを別に作る必要がなく，管理も非常に楽である．

先に述べたように，Cではアセンブラを使わなくてはならないことはあまりないのだが，もし「ちょっと使いたい」というような場合には，このインラインアセンブラ機能のほうが別個にアセンブラソースファイルを作るよりは楽でいい．

ただし，インラインアセンブラ機能は，各社のCコンパイラ間で記述方式に互換性が期待できない．たとえば，後述するがインラインアセンブラ文を識別する予約語がMS-Cでは“_asm”であるが，Turbo Cでは“asm”である．そういう人がいるかどうかわからないが，しょっちゅうコンパイラをとりかえて使うような人は注意したほうがいい．

Turbo C，MS-Cとも，基本的なインラインアセンブラの機能は同等である．8086のすべての命令コードが記述可能で，MASMのような高度なマクロこそ使えないが，インラインアセンブラではもともとあまり大きなものを書くことは少ないだろうから，不自由はないだろう．

おもしろいのは“_emit”（Turbo Cでは“__emit__”）で，ちょうどアセンブラのDB擬似命令のように，任意のデータをその位置に埋め込むことができる．MS-Cでは，1つの“_emit”で展開できるデータは1バイトだが，Turbo Cでは複数のデータを一度に，かつワード値や4バイトフルアドレスの展開もできる．Turbo Cは，インラインアセンブラにも力を入れて，それなりに強力なものに仕立てているようだ．マニアにはうれしい姿勢だろう．

図14　emit書式(MS-C)

```
_emit 0x66          ';' は不要なので注意
```

図15　emit書式(Turbo C)

```
int     c;
        ........
    __emit__(0x90, (unsigned)-1, (void far *)&c);
        ........
```


アセンブラ用語集
デバッグ　プログラムの誤りを訂正する作業．デバッガを使わず，目でプログラムの流れを追いながらする作業を机上デバッグという．

インラインアセンブラ文を識別するための予約語は，MS-Cが“_asm”，Turbo Cが“asm”である．MS-Cではインラインアセンブラ文を“{~}”で囲んでブロックにすることができる．では，このインラインアセンブラ機能を使って例題のビットカウントプログラムを書き換えてみよう．まずは，関数“bit_count”をそのままインラインアセンブラで置き換えてみる（リスト11）．

別ファイルにしていたアセンブラソース“bcasm.asm”とくらべると，ずいぶん簡単になったことがわかるだろう．入り口や出口の処理は，Cの関数としてコンパイラが自動的に生成するので書かなくてもよくなった．戻り値は，すでにaxに入っているので，そのまま終了すればいい．もちろんret命令も自動的に生成されるので不要である．

では次だ．このやりかたでは関数“bit_count”は確かに速くなったが，関数呼び出しのオーバーヘッドは依然として存在する．そこで，関数呼び出しではなくこれを呼び出す“main”のなかにこの処理を直接埋め込んでしまおう（リスト12）．ここからがインラインアセンブラの本領発揮というところだ．

ごらんのように，Cレベルで定義された変数（“result”）にもシンボル名を使って自由にアクセスできる．当然のことだが，この場合は関数の中の普通の定義として扱われるので，インラインアセンブラ記述の前後には例の入り口や出口の処理は生成されていない．そのかわり，結果は関数の戻り値として返すのではなく変数に書き込んでいる．“printf”の引数の中に直接インラインアセンブラ定義を書き込めないかと試してみたのだが，さすがにそれは許されていないようである．

同じことをTurbo Cでもやってみよう（リスト13，14）．こちらは，MS-Cのようにインラインアセンブラ記述をブロック扱いにする機能はないようなので，1行ずつ“asm”をつけなくてはならないのだが，そうたいした手間ではない．また，Turbo Cでコンパイルする場合は“-B”オプションをつけるか，ソース文中に“#pragma inline”を記述しておくこと．そうしておかなくても，インラインアセンブラ記述を見つけると，自動的に“-B”オプションが指定されたものとして最初からコンパイルをしなおしてくれるようだが，警告が出るのは楽しくないので，きちんとオプションを指定しておくことをお勧めする．

アセンブラ関数からCの関数を呼び出す

ところで，理屈がわかってみるとあたりまえのことだが，この逆もまた可能である．つまり，アセンブラ記述の関数からCの関数を呼び出すこともできる．すでにわかったようにCの関数

リスト11 BCI1.C(MS-C)

```
#include    <stdlib.h>

int      bit_count(int n)
{
    _asm{
        xor      ax,ax
        mov      bx,[bp+4]
        mov      cx,16
    _lp:
        shl      bx,1
        adc      ax,0
        loop     _lp
    }
}

void     main(int argc,char **argv)
{
int      result;
    if(argc != 2)
    {
        printf("Usage: A>BC <number>¥n");
        exit(1);
    }
    result = atoi(argv[1]);
    printf("Answer = %d¥n",bit_count(result));
    exit(0);
}
```

リスト12 BCI2.C(MS-C)

```
#include    <stdlib.h>

void     main(int argc,char **argv)
{
int      result;
    if(argc != 2)
    {
        printf("Usage: A>BC <number>¥n");
        exit(1);
    }
    result = atoi(argv[1]);
        _asm{
            xor      ax,ax
            mov      bx,result
            mov      cx,16
        _lp:
            shl      bx,1
            adc      ax,0
            loop     _lp
            mov      result,ax
        }
    printf("Answer = %d¥n",result);
    exit(0);
}
```

アセンブラ用語集
ニーモニック　2進数の集まりであるマシンコードを人間が理解し易いように簡略化された単語をニーモニックという．

というのは，われわれがいまアセンブラで記述した関数と同じシーケンスのコードをコンパイラが自動的に生成しているというだけのことである．

試しにやってみよう．まずCの“main”関数がコマンドライン引数から数値を取り出して，これを引数としてアセンブラ関数を呼び出す（リスト15)．アセンブラ関数のほうはこれを整数として半分にし，十進表示をするCの関数を呼び出す(リスト16)．うまくいくかな．

```
A>cl /c calltest.c callc.asm
A>link calltest+callc;

A>CALLTEST 2
Answer = 1

A>CALLTEST 8
Answer = 4

A>CALLTEST 9
Answer = 4
```

リスト13　BCI.C(Turbo C)

```
#include    <stdlib.h>

int     bit_count(int n)
{
    asm     xor     ax,ax
    asm     mov     bx,[bp+4]
    asm     mov     cx,16
    _lp:
    asm     shl     bx,1
    asm     adc     ax,0
    asm     loop    _lp
}

void    main(int argc,char **argv)
{
int     result;
    if(argc != 2)
    {
        printf("Usage: A>BC <number>¥n");
        exit(1);
    }
    result = atoi(argv[1]);
    printf("Answer = %d¥n",bit_count(result));
    exit(0);
}
```

リスト14　BCI2.C(Turbo C)

```
#include    <stdlib.h>

void    main(int argc,char **argv)
{
int     result;
    if(argc != 2)
    {
        printf("Usage: A>BC <number>¥n");
        exit(1);
    }
    result = atoi(argv[1]);
        asm     xor     ax,ax
        asm     mov     bx,result
        asm     mov     cx,16
        _lp:
        asm     shl     bx,1
        asm     adc     ax,0
        asm     loop    _lp
        asm     mov     result,ax
    printf("Answer = %d¥n",result);
    exit(0);
}
```

リスト15　CALLTEST.C(MS-C)

```
#include        <stdlib.h>


void    data_print(int n)
{
    printf("Answer = %d¥n",n);
}

void    main(int argc,char **argv)
{
int             result;
        if(argc != 2)
        {
                printf("Usage: A>CALLTEST <number>¥n");
                exit(1);
        }
        result = atoi(argv[1]);
    div2_print(result);
    exit(0);
}
```

リスト16　CALLC.ASM

```
_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP    _BSS, _DATA
        ASSUME DS: DGROUP, SS: DGROUP

EXTRN   _data_print:near

_TEXT      SEGMENT
        ASSUME  CS: _TEXT
PUBLIC          _div2_print
_div2_print     PROC
        shr     ax,1
        push    ax
        call    _data_print
        pop     ax
        ret
_div2_print     ENDP

_TEXT   ENDS

END
```


アセンブラ用語集
ハードウェア割り込み　CPUはINTR端子に接続されたハードウェアからの割り込み信号を受けて，割り込み処理ルーチンに制御を移す．RS 232 Cなどの制御に使われる．

うまくいっているようだ．普通こんなことはしないとは思うが，できるということを覚えておくのも悪くないだろう．「なんだかだまされてるみたい」と思った人は，Cのソースから/Faオプションでアセンブラソースを出させてトレースしてみるといい．

簡単ではあるが，以上でアセンブラ記述のCの関数の作成方法とインラインアセンブラ機能の紹介を終わる．おわかりいただけただろうか？　慣れないととっつきにくいかもしれないが，せっかくの機能であるから大いに活用していただきたい．

付録(オブジェクトモジュールダンププログラム)

オブジェクトモジュールを見やすくダンプ！

オブジェクトモジュールを少々見やすくダンプするためのツールである（リスト17）．オブジェクトファイル名を指定してこのプログラムを起動すると，レコード種別とそれに対応したレコード名，そのレコードに含まれるデータサイズとデータのダンプを出力する．ファイル名は“.obj”まで書くこと．

開発は最初LSIC-86で行ったが，特別なことはなにもしていないので，Turbo CあるいはMS-Cでもコンパイルすることができるし，動作は確認ずみである．動作を確認していてバグを見つけたというようなことは，読者は知らなくてよろしい（とほほ）．

図16　出力例(A.OBJ)

```
Code = 80 , Name = 'THEADR'
Data size =    7 Bytes
Data = 05 61 2E 61 73 6D A4  /.a.asm.

Code = 96 , Name = 'LNAMES'
Data size =   25 Bytes
Data = 00 06 44 47 52 4F 55 50 04 44 41 54 41 04 43 4F  /..DGROUP.DATA.CO
       44 45 05 53 54 41 43 4B C2  /DE.STACK.

Code = 98 , Name = 'SEGDEF'
Data size =    7 Bytes
Data = 48 00 00 03 03 01 12  /H......

Code = 98 , Name = 'SEGDEF'
Data size =    7 Bytes
Data = 74 00 00 05 01 01 E6  /t......

Code = 98 , Name = 'SEGDEF'
Data size =    7 Bytes
Data = 28 04 00 04 04 01 2C  /(......

Code = 9A , Name = 'GRPDEF'
Data size =    6 Bytes
Data = 02 FF 01 FF 02 5D  /.....]

Code = 8C , Name = 'EXTDEF'
Data size =    9 Bytes
Data = 06 42 5F 50 52 4F 43 00 90  /.B_PROC..

Code = 88 , Name = 'COMENT'
Data size =    4 Bytes
Data = 00 A2 01 D1  /....

Code = A0 , Name = 'LEDATA'
Data size =    8 Bytes
Data = 03 00 00 E8 00 00 C3 AA  /........

Code = 9C , Name = 'FIXUPP'
Data size =    6 Bytes
Data = 84 01 26 01 01 B1  /..&...

Code = 8A , Name = 'MODEND'
Data size =    2 Bytes
Data = 00 74  /.t
```

図17　出力例(B.OBJ)

```
Code = 80 , Name = 'THEADR'
Data size =    7 Bytes
Data = 05 62 2E 61 73 6D A3  /.b.asm.

Code = 96 , Name = 'LNAMES'
Data size =   25 Bytes
Data = 00 06 44 47 52 4F 55 50 04 44 41 54 41 04 43 4F  /..DGROUP.DATA.CO
       44 45 05 53 54 41 43 4B C2  /DE.STACK.

Code = 98 , Name = 'SEGDEF'
Data size =    7 Bytes
Data = 48 00 00 03 03 01 12  /H......

Code = 98 , Name = 'SEGDEF'
Data size =    7 Bytes
Data = 74 00 00 05 01 01 E6  /t......

Code = 98 , Name = 'SEGDEF'
Data size =    7 Bytes
Data = 28 02 00 04 04 01 2E  /(......

Code = 9A , Name = 'GRPDEF'
Data size =    6 Bytes
Data = 02 FF 01 FF 02 5D  /.....]

Code = 90 , Name = 'PUBDEF'
Data size =   13 Bytes
Data = 00 03 06 42 5F 50 52 4F 43 00 00 00 85  /...B_PROC....

Code = 88 , Name = 'COMENT'
Data size =    4 Bytes
Data = 00 A2 01 D1  /....

Code = A0 , Name = 'LEDATA'
Data size =    6 Bytes
Data = 03 00 00 48 C3 4C  /...H.L

Code = 8A , Name = 'MODEND'
Data size =    2 Bytes
Data = 00 74  /.t
```

アセンブラ用語集

パリティフラグ　演算の結果，下位8ビットの中でセットされているビットが偶数ならセットされ，奇数ならリセットされるフラグ．エラーチェックに使われる．

リスト17　オブジェクトモジュールダンププログラム

```c

/* ==============================================
 *
 * Object file(.obj) Dump Program
 *
 * Usage : A>OBJDUMP filename
 *
 * Copyright(C) 1991 By Justech Labo.
 *
 *==============================================
 */

#include    <stdio.h>
#include    <stdlib.h>

#define     TRUE        1
#define     FALSE       0
#define     UNKNOWN     FALSE

#define     DUMPWIDTH   16

#define     THEADR      0x80
#define     COMENT      0x88
#define     MODEND      0x8a
#define     EXTDEF      0x8c
#define     TYPDEF      0x8e
#define     PUBDEF      0x90
#define     LINNUM      0x94
#define     LNAMES      0x96
#define     SEGDEF      0x98
#define     GRPDEF      0x9a
#define     FIXUPP      0x9c
#define     LEDATA      0xa0
#define     LIDATA      0xa2
#define     COMDEF      0xb0

#define     ENDCOD      0x00

struct  OMC{
    unsigned char   OM_code;
    char            *OM_code_name;
}omc[] = {
        THEADR , "THEADR",
        COMENT , "COMENT",
        MODEND , "MODEND",
        EXTDEF , "EXTDEF",
        TYPDEF , "TYPDEF",
        PUBDEF , "PUBDEF",
        LINNUM , "LINNUM",
        LNAMES , "LNAMES",
        SEGDEF , "SEGDEF",
        GRPDEF , "GRPDEF",
        FIXUPP , "FIXUPP",
        LEDATA , "LEDATA",
        LIDATA , "LIDATA",
        COMDEF , "COMDEF",
        ENDCOD , ""
    };

FILE    *fp;

/* ==================================================================
 * ファイルをクローズして終了する。ファイルはｅｘｉｔ（）で強制クローズ
 * されるはずだが、ちょっと気持ち悪いので自分でクローズしている。
 * ==================================================================
 */

void    fclose_and_exit(int return_code)
{
    fclose(fp);
    exit(return_code);
}

/* ==========================================================
 * オブジェクトファイルから１バイト読み出す。エラーがあったら
 * この中で強制終了する。
 * ==========================================================
 */

int     get_byte_with_error_exit(void)
{
int     data;

    if((data = fgetc(fp)) == EOF)
    {
        printf("Object module error¥n");
        fclose_and_exit(1);
    }
    return data;
}

/* ====================================
 * オブジェクトレコードの名称を表示する。
 * ====================================
 */

void    name_disp(int code)
{
struct  OMC *omcp;

    for(omcp = omc ; omcp->OM_code != ENDCOD ; omcp++)
    {
        if(omcp->OM_code == code)
        {
            printf("Code = %02X , Name = '%s'¥n", code , omcp->OM_code_name);
            return;
        }
    }
    printf("Unknown code (%02X)¥n",code);
    fclose_and_exit(1);
}

/* ======================
 * レコードサイズを取得する
 * ======================
 */

int     get_body_size(void)
{
int     body_size;

        body_size  = get_byte_with_error_exit();
        body_size += get_byte_with_error_exit() << 8;
        return body_size;
}

/* ==========================
 * レコードの内容をダンプする。
 * ==========================
 */

void        body_dump()
{
int         body_size,data,count;
char        asc[DUMPWIDTH + 1];

    body_size = get_body_size();
    printf("Data size = %4d Bytes¥n",body_size);

    printf("Data = ");
    for(count = 0 ; body_size ; body_size--)
    {
        data = get_byte_with_error_exit();
        printf("%02X ",data);
        asc[count] = (data >= 0x20) && (data <= 0x7f) ? data : '.';
        count++;
        if(count == DUMPWIDTH)
        {
            asc[count] = '¥0';
            printf(" /%s¥n       ",asc);
            count = 0;
        }
    }
    if(count)
    {
        asc[count] = '¥0';
        printf(" /%s¥n",asc);
    }
}

/* ==============
 * メインルーチン。
 * ==============
 */

int     main(int argc , char **argv)
{
int     code;

    if(argc != 2)
    {
        printf("Usage : A>OBJDUMP filename¥n");
        exit(1);
    }
    fp = fopen(argv[1],"rb");
    if(fp == (FILE *)NULL)
    {
        printf("Can't open %s¥n",argv[1]);
        exit(1);
    }
    for(;;)
    {
        code = get_byte_with_error_exit();
        name_disp(code);
        body_dump();
        if(code == MODEND)
            break;
        printf("¥n¥n");
    }
    fclose_and_exit(0);
}
```

参考文献：

MS－DOSエンサイクロペディア Vol1（アスキー出版局）

ソフトウェア考現学（荻谷昌己著　CQ出版社）

Microsoft C Professional Development System Programming Guide（マイクロソフト株式会社）

Microsoft C Professional Development System Reference（マイクロソフト株式会社）

TURBO C USER'S GUIDE（Borland International）

TURBO C REFERENCE GUIDE（Borland International）

アセンブラ用語集
バイト　8ビットを1バイトといい，コンピュータでもっともよく使われる単位.



4

逆アセンブルの楽しみ

ツールとその使い方

清 十 郎

パソコン用に市販されているソフトウェアの大部分は実行形のみの販売で，そのソースが付属していることは希である．このようなプログラムをちょっと改造したいというときに威力を発揮するのが，逆アセンブルというテクニックだ．市販またはフリーソフトウェアとして配布されているソースジェネレータを紹介するとともに，具体的な逆アセンブルのテクニックを解説する．

ソースがなくても大丈夫!?

◆ソースジェネレータはプログラマの味方

古い話で恐縮ですが……，私がパソコンに関わりはじめたころは，ROM BASIC（！）が全盛で，マシン語モニタ（デバッガではない）といえば，貧弱なMONコマンドがあるだけでした．アセンブラにしても，市販のものは高くてとても手が出なかったので，開発環境を自分たちで作り出さねばならない状況でした．そんなときに，高機能のマクロアセンブラとデバッガを含んだMS-DOS Ver.2.11が，たった1万円で売りだされたのです．

実はそのとき，8ビット機で主流だったCP/Mも同じ値段だったのですが，やはりMASM&DEBUGの環境にくらべると，とても不便なものに見えて，しだいに使わなくなってしまいました．そのDEBUGも，SYMDEBからCodeViewへと進化して，ソースレベル（！）のデバッグができるようになりましたが，こんどは肥大化し過ぎて，かえって使いづらくなってしまいました．それで，いまはTASM（Turbo Assembler）とTD（Turbo Debugger）です(^_^)；

そのTDで，.COM形式のプログラムをソースレベルデバッグする方法をご存じでしょうか？　.EXEの場合は，TASMで-ZIを，TLINKで/vオプションを指定するだけでいいのですが，.COMにするときに，TLINKに/tオプションを追加しただけのものでTDを起動すると，“シンボルテーブルがありません”といって叱られてしまいます．いろいろオプションをかえてもダメです．なぜなら，/tオプションで.COMにすると，シンボル情報を切り捨ててしまうからです．シンボル情報を生かすには，TDのユーティリティの1つ，TDSTRIPで.COMにしなければなりません．

```
A:¥>TDSTRIP -s -c filename
```

とすることで，シンボルテーブルのついた.EXEファイルから.COMと.TDSシンボルファイルが変換されます．これでTDを起動すれば，めでたくソースレベルのデバッグができるようになるわけです．

しかし，ソースがない場合はどうでしょう．昔作ったプログラムでソースが見つからないとか，ディスクがクラッシュしてソースがなくなった（；_；）とか，あるいは，既成のプログラムを使いやすいように直したい（最近はカスタマイズできるソフトも増えましたが，まだまだ少数です）ということもよくあります．そんな場合は，進化の過程をさかのぼってDEBUGレベルに戻らなくてはならないのでしょうか？　ご心配はいりません（^_^）．そんなときにこそ，これから紹介するソースジェネレータと呼ばれるツールがお役に立つことでしょう．

なお，紹介にあたって各処理系の違いを顕在化させるために，かんたんなプログラム（リスト1）を作成しました．同じプログラムが，どのようにジェネレートされるか確かめてください．

リスト1　KZAT_ORG.ASM

```
;-------------------------------------------;
;          数当てプログラム        91/02/16  ;
; KZAT Version 1.00 (C) 1991   By Sey-ju-row ;
;-------------------------------------------;

code            segment
                assume  cs:code,ds:code

                ORG 100h
start:

        mov     ah,2ch
        int     21h             ;時刻を得る

        mov     bl,dh           ;秒を保存しておく
        and     bl,0fh
        mov     cx,lop_cnt
loop0:
        mov     dx,offset in_msg
        mov     ah,9
        int     21h             ;文字列の出力

        mov     ah,1
        int     21h             ;1文字入力

        call    var
        cmp     bl,al
        je      ok              ;同じ数か？
        mov     dx,offset ng_msg
        mov     ah,9
        int     21h

        loop    loop0

        mov     dx,offset end_msg
        mov     ah,9
        int     21h

        jmp     short exit
ok:
        mov     dx,offset ok_msg
        mov     ah,9
        int     21h
exit:
        mov     ah,4ch
        int     21h


;====== subroutine ======

var     proc    near            ;1文字を数字に変換
        call    con_cap
        sub     al,'0'
        jb      not_num
        cmp     al,9
        jbe     num
```

アセンブラ用語集
汎用レジスタ　AX，BX，CX，DXの4つのレジスタを汎用レジスタという．レジスタ長は16ビット以外に8ビットとしても利用できる．

```
        sub     al,7
        cmp     al,10h
        jb      num
not_num:
        stc
        jmp     short var_ret
num:
        clc
var_ret:
        ret
var     endp

con_cap proc    near                ;小文字を大文字に変換
        cmp     al,'a'
        jb      cncp_ret
        cmp     al,'z'
        ja      cncp_ret
        and     al,0dfh             ;MASK THE 5th BIT
cncp_ret:
        ret
con_cap endp


lop_cnt dw      4

in_msg  db      0dh,0ah,9,'No. = $'

ng_msg  db      7,9,9,'違いますョ',0dh,0ah,'$'

ok_msg  db      8 dup (7),9,'ＯＫ！',0dh,0ah,'$'

end_msg db      0dh,0ah,9,9,9,'残念でした^^;',0dh,0ah,'$'

        ;
code    ends
        end     start
```

SOURCER

超人気のIBM PC用逆アセンブラ

SOURCERは，1988年9月にインターフェース誌に紹介されてから超人気となり，IBMユーザーなら知らぬものがないほどのソフトです．Ver.3には，漢字を扱える-jオプションと，IBM特有のグラフィックキャラクタを使用しないという-n1オプションが追加されたので，SIMなどのIBM PCシミュレータを使わなくてもPC-9801上で動くようになりました．

日本では，(株)サザンパシフィックが輸入代理店となっています．

パッケージには，5"2Dのディスクと140ページのマニュアル，そしてコマンドオプションの一覧表などが納められています．ディスクの内容を図1に示します．このうち実行に必要なのは，SR.EXEとSDEFAULT.DEFくらいです．マニュアルには，README.BATでインストールするように書いてありますが（PC-9801では使えない）単純にコピーすればOKです．

また，BIOSP.EXEは合法的にIBM PCのBIOSのソースコードを得るツールなので，PC-9801で動作させても意味がありません．

-n1オプションによりMS-DOS汎用で動きますが，モノクロですし，SOURCER独特のロゴも美しくありません．かといって，SIMなどはパッチファイルが必要だったりしてコンピュータの知識があまりないものにとっては使い方がむずかしいのですが，そんな人にうってつけのソフトが，The BASICの1990年5月号で紹介されていました．LXIBMというものです．このプログラムはほとんどインスタントで使えるので便利です．フリーソフトウェアとして，NIFTY-Serve FIBM LIB9などに登録されています．

これを使ってSR.EXEを動かす方法を図2に示します．このようにRUNSR.BATを作成しておけば，次のように入力することで，IBM PC上のようにフルカラーのSOURCERが使えます．

A:¥>RUNSR　file_name.type

SOURCERの使用法

起動すると，EMSが使えるかどうか

図1　SOURCER製品ディスクの内容

README	BAT	2151	90-12-12	8:00
SR	EXE	150661	90-12-12	8:00
SDEFAULT	DEF	5887	90-12-12	8:00
TESTYN	EXE	846	88-03-26	15:23
TESTYN	DEF	1695	90-12-12	8:00
TESTYN	REM	5273	90-12-12	8:00
SAMPLE1	DEF	17312	90-12-12	8:00
SAMPLE2	DEF	1105	90-12-12	8:00
INFOSRB	EXE	25589	90-12-12	8:00
LST2ASM	EXE	14038	90-12-12	8:00
RUNBIOSP	BAT	460	90-12-12	8:00
BIOSP	EXE	52791	90-12-12	8:00
BIOS	REM	77834	90-12-12	8:00

図2　LXIBMを用いてSR.EXEを動かす方法

```
A:¥>IO2NOP SR.EXE SRX.EXE          ←パッチをかけた実行型ファイルを作る．
A:¥>COPY CON > RUNSR.BAT           ←リダイレクトで起動用バッチを作る．
LXIBM SRX.EXE %1 -j
^Z                                 ←[Ctrl]+[Z]または[F･10]
 1 個のファイルをコピーしました．
A:¥>
```

図3　SOURCERの定義ファイルの書式

```
seg:off        type&options    label    ;comment
```

表1　SOURCERコマンドリファレンス

キー	機能	解説
[F·1]	ヘルプ	コマンド，オプションの解説ウインドウを表示する．
[ESC]	終了	いつでもDOSに戻る．
[A]	解析オプション	デフォルトの解析法と，出力オプションを変更します．サブウインドウが開き，各オプションを入力するごとに，on(大文字)/off（小文字）が切り替わる．
[B]	開始アドレス	プロセスの開始アドレスを変更する（RAM，ROMの解析用）．
[C]	コード形式	COM，EXE，デバイスドライバなどを指定する．
[D]	ドライブ	ファイルを出力するドライブを選択．
[E]	終了アドレス	プロセスの終了アドレスを変更する（RAM，ROMの解析用）．
[F]	ファイル形式	出力ファイルをリスト形式とするか，アセンブラのソースとするかを選択．
[G]	Go	入力ファイルまたはアドレスレンジ内の解析を開始する．
[H]	ヘッダタイトル	リストの各ページの先頭に入れるタイトル名を登録する．
[I]	入力ファイル名	処理をするファイル名を変更する．ドライブ名：パス名での指定も可能．
[L]	ラベルタイプ	ラベル名のタイプを選択する．10進数（data_6）やセグメント：オフセット（d_7B23_08AB）など4種類．
[M]	数値演算プロセッサ	80x87の命令をデコードするかを選択する．
[O]	出力ファイル名	出力をするファイル名を変更する．ドライブ名：パス名での指定も可能．
[P]	パス	処理パスの回数を選択します．2から9まであり，パスをかさねるほど解析精度は向上する．
[Q]	終了	いつでもDOSに戻る．
[R]	REM	自動的に入る注釈文のタイプを選択する．1から8までの8種類．
[S]	セグメントon/off	リストファイルのアドレス表示部にセグメントを入れるかどうかを選択する．
[T]	ターゲットアセンブラ	MASM，TASM，OPTASMの内，いずれを対象とするかを選択する．
[U]	プロセッサ	8086から80486の内，いずれのプロセッサの命令をデコードするかを選択する．
[W]	ワードスタイル	出力ファイルの文字を，大文字とするか小文字にするかを選択する．
[X]	クロスリファレンスon/off	5か，それ以上のパスのリストにインラインクロスリファレンスを入れるかどうかを切り替える．

調べ，使えるようなら初期化します．これは-eオプションで抑止できます（そのはずなのですが，PC-286NFで2MBのRAMの半分をRAMディスクとして使っているため，このオプションを指定してみたのですが，効きめはありませんでした）．

次に，メインメニューの画面になります．ここで入力できるコマンドリファレンスを表1に示します．これらのうち，いくつかの項目から選択するものは，そのキーを押すごとに切りかわります．また，表示の仕方を変えるものは，画面上の例を変化させて表示します．

ここでの設定は，自動的に作成されるfile_name.SDFに登録されます．これをfile_name.DEFにリネームしておくと，file_nameだけの指定で前回と同じ環境でSOURCERを起動することができます．

リスト2　KZAT_SR.ASM

```
PAGE  59,132

;##########################################################################
;##                                                                      ##
;##                              KZAT                                    ##
;##                                                                      ##
;##      Created:   16-Feb-91                                            ##
;##      Version:                                                        ##
;##      Passes:    5          Analysis Options on: OQRS                 ##
;##                                                                      ##
;##                                                                      ##
;##########################################################################

seg_a           segment byte public
                assume  cs:seg_a, ds:seg_a


                org     100h

KZAT            proc    far

start:
                mov     ah,2Ch                  ; '.'
                int     21h                     ; DOS Services  ah=function 2Ch
                                                ;  get time, cx=hrs/min, dh=sec
                mov     bl,dh
                and     bl,0Fh
                mov     cx,data_1               ; (5D96:015D=4)

locloop_1:                                      ;  xref 5D96:0126
                mov     dx,offset data_2        ; (5D96:015F=0Dh)
                mov     ah,9
                int     21h                     ; DOS Services  ah=function 09h
                                                ;  display char string at ds:dx
                mov     ah,1
                int     21h                     ; DOS Services  ah=function 01h
                                                ;  get keybd char al, with echo
                call    sub_1                   ; (013C)
                cmp     bl,al
                je      loc_2                   ; Jump if equal
                mov     dx,offset data_3        ; (5D96:0169=7)
                mov     ah,9
                int     21h                     ; DOS Services  ah=function 09h
                                                ;  display char string at ds:dx
                loop    locloop_1               ; Loop if cx > 0

                mov     dx,offset data_5        ; (5D96:018B=0Dh)
                mov     ah,9
                int     21h                     ; DOS Services  ah=function 09h
                                                ;  display char string at ds:dx
                jmp     short loc_3             ; (0138)
```


アセンブラ用語集
ファンクションコール　MS-DOSシステムコールのひとつで，INT 21 Hを指す．AHレジスタに機能（ファンクション）番号を入れ呼び出すことで多くの機能を実現する．

設定が終わったら，Gコマンドを押して処理を開始します．なお，解析が終了した時点でDOSに戻ります．

KZAT.COMをSOURCERで解析して得られたアセンブラソースがリスト2です．じつにほれぼれとする美しいソースがオートで作られています．データ部の分離も完璧ですし，漢字も“OK”のあとの“!”が分割されている以外はちゃんと表示されています．あと，8個繰り返している7がdupでまとめられていれば，いうことなしなのですが……．

KZATは短いプログラムですし，ややっこしいことはなにもしていないのでよいのですが，実際はオートで完璧に解析されることはまれで，やはり定義ファイルをエディットしてオートの解析を助けてやる必要があります．

SOURCERの定義ファイルは自動的に作成される“.SDF”を“.DEF”にリネームして使います（図3）．

KZAT.DEFを，リスト3に示します．SOURCERの定義ファイルのよいところはラベルの属性やデータの種類，ユーザー定義ラベル名などを1行に指定できることです．タイプやオプションは50種類くらいあり，覚えるのがたいへんですが，まあ，ほとんどはオートで指定してくれます．

いずれにせよ，完成度が高いうえに，いまだに成長を続けているというエネルギッシュなプログラムです．これと，インターフェース誌の1989年2月号で紹介されたASMFLOW（現在はASMtoolと改名．アセンブラソースのフローチャート生成とソースコードの解析をするツール）や，同じくインターフェース誌1991年2月号に紹介されていたOBJASM（OBJファイルからアセンブラソースに戻すツール）などがそろえば，陳腐な表現ですがまさ

```
loc_2:                                          ;  xref 5D96:011D
                mov     dx,offset data_4        ; (5D96:0179=7)
                mov     ah,9
                int     21h                     ; DOS Services  ah=function 09h
                                                ;  display char string at ds:dx
loc_3:                                          ;  xref 5D96:012F
                mov     ah,4Ch                  ; 'L'
                int     21h                     ; DOS Services  ah=function 4Ch
                                                ;  terminate with al=return code

KZAT            endp

;##########################################################################
;                               SUBROUTINE
;
;
;          Called from:   5D96:0118
;##########################################################################

sub_1           proc    near
                call    sub_2                   ; (0152)
                sub     al,30h                  ; '0'
                jc      loc_4                   ; Jump if carry Set
                cmp     al,9
                jbe     loc_5                   ; Jump if below or =
                sub     al,7
                cmp     al,10h
                jb      loc_5                   ; Jump if below
loc_4:                                          ;  xref 5D96:0141
                stc                             ; Set carry flag
                jmp     short loc_ret_6         ; (0151)
loc_5:                                          ;  xref 5D96:0145, 014B
                clc                             ; Clear carry flag

loc_ret_6:                                      ;  xref 5D96:014E
                retn
sub_1           endp

;##########################################################################
;                               SUBROUTINE
;
;
;          Called from:   5D96:013C
;##########################################################################

sub_2           proc    near
                cmp     al,61h                  ; 'a'
                jb      loc_ret_7               ; Jump if below
                cmp     al,7Ah                  ; 'z'
                ja      loc_ret_7               ; Jump if above
                and     al,0DFh

loc_ret_7:                                      ;  xref 5D96:0154, 0158
                retn
sub_2           endp

data_1          dw      4                       ;  xref 5D96:0109
data_2          db      0Dh, 0Ah, 9, 'No. = $'  ;  xref 5D96:010D, 0112
data_3          db      7, 9, 9, '違いますョ', 0Dh, 0Ah, '$'   ;  xref 5D96:011F, 0124
data_4          db      7, 7, 7, 7, 7, 7, 7, 7, 9, 'ＯＫ'      ;  xref 5D96:0131, 0136
                db      '!', 0Dh, 0Ah, '$'
data_5          db      0Dh, 0Ah, 9, 9, 9, '残念でした^^;'     ;  xref 5D96:0128, 012D
                db      0Dh, 0Ah, '$'

seg_a           ends


                end     start
==================== CROSS REFERENCE - KEY ENTRY POINTS ===================

    seg:off     type       label
   ---- ----    ----    --------------
  5D96:0100     far      start

 ================== Interrupt Usage Synopsis ==================

        Interrupt 21h :  get keybd char al, with echo
        Interrupt 21h :  display char string at ds:dx
        Interrupt 21h :  get time, cx=hrs/min, dh=sec
        Interrupt 21h :  terminate with al=return code

 ================== I/O Port Usage Synopsis  ==================

        No I/O ports used.
```

リスト3　KZAT_SR.DEF

```
                ############################################
                         SOURCER DEFINITION FILE
                ############################################

 ================== Section 1: CONTROL INFORMATION   ==================

Analysis Options = a b c d e f g h i j k l m n O p Q R S t u v w x y z
uP               = 80286
File format      = ASM
Label type       = Decimal
Remarks          = All
Target Assembler = TASM-2.0
Input filename   = KZAT.COM
Code style       = Com
Drive (output)   = A
Output filename  = KZAT.ASM
Passes           = 5
Xref             = ON
```

アセンブラ用語集
フラグレジスタ　演算命令の実行結果などを1ビット単位で格納する特別なレジスタ．

に鬼に金棒！^^;;と，いったところです．

製品名：SOURCER
SOURCER with
BIOS PRE-PROCESSOR V3.13
8088 to 80486
Commenting Disassembler
価格：169.95ドル
連絡先：V COMMUNICATIONS,INC.
4320 Stevens Creek Blvd., Suite 275
San Jose, CA 95129
PHONE:408-296-4224
FAX:408-296-4441
輸入代理店：㈱サザンパシフィック
〒220 横浜市西区南幸2-16-20 三和横浜ビル
電話：045-314-9514

```
================== Section 2: RANGE DEFINITION       ==================

==== Segments ==========================

 begin      end      default       seg   seg     seg
seg:off     off     ds      es     type  size    value
-------     ---     ----    ----   ----- -----   -----
seg_a:0100  019F    seg_a  seg_a   auto  use16   ; 5D96

================== Section 3: REFERENCE DEFINITIONS ==================

==== Subroutines ======================
  seg:off   type & options     label            comments
  -------   --------------     --------------   --------------
seg_a:013C  sub                 var              ; sub_1
seg_a:0152  sub                 con_cap          ; sub_2

==== Locations ========================
  seg:off   type & options     label            comments
  -------   --------------     --------------   --------------
seg_a:0100  loc, far           start
seg_a:010D  forced, od seg_a
seg_a:010D  loc                 loop0            ; locloop_1
seg_a:011F  forced, od seg_a
seg_a:0128  forced, od seg_a
seg_a:0131  forced, od seg_a
seg_a:0131  loc                 ok               ; loc_2
seg_a:0138  loc                 exit             ; loc_3
seg_a:014D  loc                 not_num          ; loc_4
seg_a:0150  loc                 num              ; loc_5
seg_a:0151  loc                 var_ret          ; loc_ret_6
seg_a:015C  loc                 cncp_ret         ; loc_ret_7

==== Data Items =======================
  seg:off   type & options     label            comments
  -------   --------------     --------------   --------------
seg_a:015D  dw                  lop_cnt          ; data_1
seg_a:015F  da, r 0A            in_msg           ; data_2
seg_a:0169  da, r 10            ng_msg           ; data_3
seg_a:0179  db, dup,r 8         ok_msg           ; data_4
seg_a:0181  da, r 4
seg_a:018B  da, r 15            end_msg          ; data_5
```

輪廻286

SOURCERに次ぐ実力を誇る

以前『IQ輪廻』という名前で出ていたものの286対応版です．パッケージには2HDのディスクと60ページのマニュアルが納められています．ディスクにはRIN.EXEしか入っていません（図4）．

マニュアルによれば，「輪廻286の自動出力したファイルを再度アセンブルし実行型ファイルへ戻してももとの状態と同じように動作するほど強力なため『輪廻286』と名付けました」ということです．実際は，ある程度訂正しないと使えるソースにはなりませんが，解析力自体はSOURCERに次ぐほどの実力があります．ただ，残念なのは定義ファイルによるきめ細かな設定ができないという点と，漢字データに対応していないという点です．

輪廻286の使用法

輪廻286の使い方を図5に示します．

はじめは自動モードを選択して起動してください．半自動モードでは，セグメンへの入力や不確定オペランドがあるごとに止まって数値の入力を求めてきますので，なかなか先に進みません．

どちらのモードでも，起動するとエントリポイントの入力を求めてきます．これは各ルーチンへの飛び込みアドレスです．エントリポイントが多い場合は，リダイレクトで入力するように勧めています．もちろん，最初はわかりませんのでRETのみを入力します．

解析を開始すると，図6のようなメッセージがドドッと表示されて驚かされます．

これもリダイレクトでファイル化しておくほうがよいでしょう．これらのメッセージを参考に正しい値がわかれば，こんどは半自動モードで起動して順次入力していくわけです．しかし，

図4　輪廻286製品ディスクの内容

```
RIN     .EXE    138759  89-03-27   1:02
```

図5　輪廻286の使用法

書式：RIN 入力ファイル名 [出力ファイル名]　[-d]　[-p]　[-o]

-d……自動モードで起動
-p……分岐命令のインデント設定
-o……org命令のインデント設定

アセンブラ用語集
プロシージャ　プログラム本体とは別に定義したサブルーチンを，とくにプロシージャという．C言語でいえば関数にあたる．

これは実際やるとたいへんな作業です．やはり，定義ファイルで指定できるようにしてもらいたいところです．

また，ルーチンの入口アドレスのエントリでは，いったい何か所入力すれば十分なのかわからないので，データ領域を指定できるようにしたほうが効率的だと思います（エントリポイントを20か所ほど指定してみたところ，解析に失敗して止まらなくなってしまいました．これは，バグなのでしょうか？）．

KZAT.COMを輪廻286で解析して得られたアセンブラソースがリスト4です．

図6　輪廻286のメッセージ

```
seg FFF0:02D5  オペランドが不確定
seg FFF0:03A6  オペランド絶対値 = 0060H
pc  FFF0:0586  未定義命令
pc  FFF0:077E  トランスファベクトルは推定
DS  FFF0:0A45  セグメント・オーバーライドが不確定
               直前のASSUME DS:をチェックしてください
```

製品名：輪廻286
Copyright 1988 JBSoftware/MICRODATA V1.02
80286用　自動思考型ソースジェネレータ
日米合体ソフト
価格：32,000円
連絡先：（株）マイクロデータ
〒169 東京都新宿区高田馬場1－17－8
電話：03-3232-9801

リスト4　KZAT_RIN.ASM

```
        .286p
        ;far return macro
;uncomment if your assembler does not handle the 'retf' instruction
;retf   macro   vlu
;       ifb     <vlu>
;       db      0cbh
;       else
;       db      0cah
;       dw      vlu
;       endif
;       endm


        ;jump near pointer macro
jmpn    macro   label
        db      0e9h
        dw      offset label - $ - 2
        endm

C1SEG   SEGMENT
        ASSUME cs:C1SEG,ds:C1SEG,es:C1SEG
        org     100H
C1F100H LABEL FAR
        mov     ah,2CH
        int     21H
        mov     bl,dh
        and     bl,0FH
        mov     cx,C1W15DH
C1N10DH:
        mov     dx,OFFSET C1B15FH
        mov     ah,9H
        int     21H
        mov     ah,1H
        int     21H
        call    C1N13CH
        cmp     bl,al
        je      C1N131H
        mov     dx,OFFSET C1B169H
        mov     ah,9H
        int     21H
        loop    C1N10DH
        mov     dx,OFFSET C1B18BH
        mov     ah,9H
        int     21H
        jmp     SHORT C1N138H
C1N131H:
        mov     dx,OFFSET C1B179H
        mov     ah,9H
        int     21H
C1N138H:
        mov     ah,4CH
        int     21H
C1N13CH:
        call    C1N152H
        sub     al,30H
        jb      C1N14DH
        cmp     al,9H
        jbe     C1N150H
        sub     al,7H
        cmp     al,10H
        jb      C1N150H
C1N14DH:
        stc
        jmp     SHORT C1N151H
C1N150H:
        clc
C1N151H:
        ret
C1N152H:
        cmp     al,61H
        jb      C1N15CH
        cmp     al,7AH
        ja      C1N15CH
        and     al,-21H
C1N15CH:
        ret
C1W15DH LABEL WORD
        dw      4H
C1B15FH LABEL BYTE
        db      0DH,0AH,9H,'No.  = $'
C1B169H LABEL BYTE
        db      7H,9H,9H,88H,0E1H,82H,0A2H,82H,0DCH,82H,0B7H,83H,87H,0DH
        db      0AH,'$'
C1B179H LABEL BYTE
        db      7H,7H,7H,7H,7H,7H,7H,7H,9H,82H,'n',82H,'j',81H,'I',0DH
        db      0AH,'$'
C1B18BH LABEL BYTE
        db      0DH,0AH,9H,9H,9H,8EH,'c',94H,'O',82H,0C5H,82H,0B5H,82H
        db      0BDH,'^^;',0DH,0AH,'$'
C1SEG   ENDS
        END     C1F100H
```

SYMGEN
◆フリーソフトウェアのソースジェネレータ

筆者の作ったソースジェネレータです．原形は3～4年前からあったのですが，NETに掲載したのが昨年の9月ころでした．あまり使用されないだろうという作者の思いに反して意外に反響があり，さまざまな要望が寄せられ機能強化を繰り返してきました．

NIFTY-ServeのFGALBTの皆さんに，この場を借りてお礼申し上げます．反響が鎮静化したいまも，暇さえあれば懲りずにバージョンアップを続けております^^;

このプログラムは，おもに“COM”形式のプログラムを対象にしています．また，逆アセンブラはSYMDEBのものを使いますので，パスの通ったディ

アセンブラ用語集
ヘッダファイル　ソースプログラムに関連した関数・疑似命令，マクロなどヘッダ部分を収録したファイル．

リスト5　　KZAT_SG.ASM

```
;KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
;
;                               KZAT
;
;                       Original:       1991-02-16
;                       Generate:       1991-02-18
;                       Address :       0100-01a0
;
;KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK

.286p


code    segment
        assume  cs:code,ds:code

        ORG     0100h
start:
        mov     ah,2ch                          ;','
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=2c00
                                                ;時刻の取得 out>CX=時分 DX=秒
        mov     bl,dh
        and     bl,0fh
        mov     cx,w_015d
l_010d:
        mov     dx,offset b_015f
        mov     ah,09h
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=0900
                                                ;文字列のｽｸﾘｰﾝ出力 in<DS:DX=文字列
        mov     ah,01h
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=0100
                                                ;ｷｰﾎﾞｰﾄﾞ入力待ちｴｺｰあり out>AL=入力文字
        call    c_013c
        cmp     bl,al
        jz      j_0131
        mov     dx,offset b_0169
        mov     ah,09h
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=0900
                                                ;文字列のｽｸﾘｰﾝ出力 in<DS:DX=文字列
        loop    l_010d
        mov     dx,offset b_018b
        mov     ah,09h
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=0900
                                                ;文字列のｽｸﾘｰﾝ出力 in<DS:DX=文字列
        jmp     short j_0138
j_0131:
        mov     dx,offset b_0179
        mov     ah,09h
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=0900
                                                ;文字列のｽｸﾘｰﾝ出力 in<DS:DX=文字列
j_0138:
        mov     ah,4ch                          ;'L'
        int     21h                             ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=4c00
                                                ;ﾌﾟﾛｾｽの終了 in<AL=ﾘﾀｰﾝ･ｺｰﾄﾞ

        public  c_013c
c_013c  proc    near
        call    c_0152
        sub     al,30h                          ;'0'
        jb      j_014d
        cmp     al,09h
        jbe     j_0150
        sub     al,07h
        cmp     al,10h
        jb      j_0150
j_014d:
        stc
        jmp     short j_0151
j_0150:
        clc
j_0151:
        ret
c_013c  endp

        public  c_0152
c_0152  proc    near
        cmp     al,61h                          ;'a'
        jb      j_015c
        cmp     al,7ah                          ;'z'
        ja      j_015c
        and     al,0dfh
j_015c:
        ret
c_0152  endp


w_015d  dw      0004h

b_015f  db      0dh,0ah,9,'No.  = $'

b_0169  db      7,9,9,'違いますョ',0dh,0ah,'$'

b_0179  db      8 dup (7),9,'ＯＫ！',0dh,0ah,'$'

b_018b  db      0dh,0ah,9,9,9,'残念でした^^;',0dh,0ah,'$',6


        ;
code    ends

        end     start
```

リスト6　　KZAT_SG.ENT

```
-
T -
M2
R1
P3
;
015d 01a0
;
U 013C VAR
U 0152 CON_CAP
U 010D LOOP0
;
U 0100 START
U 0131 OK
U 0138 EXIT
U 014D NOT_NUM
U 0150 NUM
U 0151 VAR_RET
U 015C CNCP_RET
;
U 015F IN_MSG
U 0169 NG_MSG
U 0179 OK_MSG
U 018B END_MSG
;
U 015D W LOP_CNT
```

図7　SGEN131.LZHの内容

```
SGEN    .BAT     922  91-02-12  21:25
CMDSET  .COM    2225  91-02-11   9:40
SYMGEN  .COM   60367  91-02-23  19:49
SYMGEN  .MAN   25681  91-02-23  19:42
SAMPLE  .COM    2104  91-01-01   0:49
SAMPLE  .SMP    2353  91-02-10  21:38
SYMGEN  .DOC   10664  91-02-23  19:40
```

図8　SYMGENの使用法

書式：SGEN ファイル名　[/C]　[/D]　[/E]
　　　(ファイル名に拡張子は不要)

/C……シンボルラベルがどのアドレスで使われているかという情報をリストアップした，クロスリファレンスファイルを出力する．拡張子は“CRF”．

/D……デフォルトでは，SYMDEBが作成する逆アセンブラファイルは，ディスクの容量を考えて削除しているが，このオプションで残すことができる．拡張子は“DIS”．SGEN.BATは“DIS”ファイルがすでに存在する場合は新たに作成しない．

/E……このオプションを指定すると，データ領域の登録ができる．これによって，不要なラベルの削除ができ，より正確なジェネレートが可能．
　例：0501　052D　　;データ領域を示す
　　　052E　0562 W　;ワードデータ領域を示す

※データ領域の登録はアドレスの昇順に入力する．
※ワードデータ領域ではラベル値と照合して，一致するデータをラベル名で表示．
※これらは定義ファイルから入力するようにできる．拡張子は“ENT”．SGEN.BATは，/Eオプションが指定され，かつ拡張子“ENT”のファイルが存在すれば，自動的にこれから入力するようになっている．

この/Eオプションには，以下のサブオプションがある．

-……“ENT”ファイルを自動更新しない．

※SYMGEN.COMは“ENT”ファイルが存在しない場合は，“ENT”ファイルを自動的に作成し，存在するときにはこの指定がない限り新しいデータ領域と，I/O/C/Jサブオプションの自動更新をする．このとき，ユーザーが定義したC/J/L/B/Wの任意の属性のラベルの登録とUサブオプションは失われる．

T……タイトルの装飾文字を指定する．
　例：T %
　※この指定がない場合はファイル名の先頭の1文字を使用．

アセンブラ用語集
ベースレジスタ　BXレジスタを指し，間接アドレス指定時にアドレスレジスタ，ベースレジスタとして使用することができる．

レクトリにSYMDEBを置いておいてください．

図7に配布している圧縮ファイルの中身を示します．実行に必要なのはSGEN.BAT，CMDSET.COM，SYMGEN.COMです．

SYMGENの使用法

SYMGENの使い方を図8に示します．

KZAT.COMをSYMGENで解析して得られたアセンブラソースがリスト5です．また，定義ファイルKZAT.ENTをリスト6に示します．

プログラム名：SYMGEN
（フリーソフトウェア）
SYMBOLIC-DEBUGGER'S
DISASSEMBLER
SYMBOL LABEL GENERATOR
Ver.1.31
Author：QGA02240 清十郎（川原崎由夫）
入手先：Network Information Forum
NIFTY-Serve
(FGALST　LIB4，FLABO LIB3)

M……INT命令のメッセージの書き込みを制御する．
例：M0　(INT命令にREM文をつけない)
M1　(INT命令の種類のみをREM文でつける)
M2　(ファンクションの内容まで詳解したREM文もつける：デフォルト)

R……ワード即値入力REMの書き込みを制御する．
例：R0　(ワード即値入力命令にREM文をつけない)
R1　(ワード即値入力命令に符号つき10進数とアスキーまたは漢字キャラクタのREM文をつける：デフォルト)
RJ　(漢字キャラクタのコードをJISとする)

P……パブリック／プロシージャ宣言文を制御する．
例：P0　(パブリック／プロシージャ宣言文をつけない)
P1　(パブリック宣言文のみをつける)
P2　(パブリック／プロシージャ宣言文をつける)
P3　(プロシージャ宣言文のみをつける：デフォルト)

I……指定アドレスからはじまる命令のオペランドを強制的に，即値(イミディエイト)とみなす．
例：I 146
※BX，DX，SI，DI，BPレジスタに対する即値の入力はすべてラベルに対するオフセット値であるとしているが，これはデータである可能性もあるので，解析時に判断してほしい．ひとつの目安としては，プログラム領域外のラベルはデータであると思ってよいでしょう．これらのラベルについては，ORG疑似命令で対処してあるため，そのままアセンブラを通しても，正しいコードを出力するが，自動的に作成される"ENT"ファイルにスタートアドレス前とエンドアドレス後のラベルが作られたアドレスを，このIサブオプションとして書き出す．

O……指定アドレスからはじまる命令のオペランドを強制的に，オフセット値とみなす．
例：O 115
※AX，CXとセグメントレジスタに対する即値の入力は，すべて即値であるとしている．

C・J・L・B・W ……指定アドレスに任意の属性のラベルを登録する．
例：C 06D1

U……指定アドレスのラベルをユーザー定義の文字列と置き替える．
例：U　103　END_PROSSECE
※ワードラベルのみ，アドレスのあとにWを挿入する．
例：U　77E W TMPTBL

ASMGEN
シェアウェア版
Assembly Code Generator

ASMGENに関しては，最新のニュースを知らないので詳しくは触れませんが，SOURCERのユーザー登録カードのアンケートの中に，ほかに使っている逆アセンブラとして名前が出ていたので，米国ではバージョンアップしている可能性が高いようです．また，NIFTY-Serve FGALBT会議室11番【特設会場】MDe徹底活用講座の#34-#36に，Seyさんが訳してくださったASMGEN邦訳マニュアルがありますので興味のある方はご覧ください．

ASMGENの使用法

ASMGENの使い方を図9に示します．

これを使って，KZAT.COMから得られたアセンブラソースはリスト7です．はじめのほうにズラズラとEQU疑似命令がありますが，これは即値の入力をすべてラベルとしているためです．1バイトの即値までラベルにすることはないし，アキュムレータ(AX)やカウントレジスタ(CX)ははずすべきだと思います．

ASMGENの定義ファイルは".SEQ"です．自動的には作成されませんが，ファイルが存在すれば自動的に読み込みます．書式はSOURCERの".DEF"とよく似ています(図10)．

このことから，ASMGENがSOURCERの原型であろうことが想像されます．KZAT.SEQをリスト8に示します．

アセンブラ用語集
ポインタ　ストリングやメモリブロックなど，直接データではなく，データを格納するアドレスを指す．

プログラム名：ASMGEN（シェアウェア）
The IBM Personal Computer
Assembly Code Generator

By J. Gersbach BTVVMLAB(U6081)
Public Domain Material
Version 2.01-Enhancements

By J. Damke BOEVM1(DAMKE)
Uploaded to PCanada by Mark Magner
November 23, 1983

リスト7　KZAT_AG.ASM

```
TITLE   KZAT    2-16-91 [2-16-91]

RET_NEAR        MACRO
DB      0C3H
ENDM
;
.RADIX  16
LF      EQU     0AH
CR      EQU     0DH
;
;INITIAL VALUES :        CS:IP   0000:0100
;                        SS:SP   0000:FFFF
                                                  ;L0000S   L015D DI
S0000   SEGMENT
        ASSUME DS:S0000, SS:S0000 ,CS:S0000 ,ES:S0000
        ORG     $+0100H
L0000   EQU     $-100
                                                  ;L0001    L0114 DI
L0001   EQU     $-0FF
                                                  ;L0007    L0147 DI
L0007   EQU     $-0F9
                                                  ;L0009    L0110 DI  L0122 DI  L0
12B DI  L0134 DI  L0143 DI
L0009   EQU     $-0F7
                                                  ;L000D    L0186 DI
L000D   EQU     $-0F3
                                                  ;L000F    L0106 DI
L000F   EQU     $-0F1
                                                  ;L0010    L0149 DI
L0010   EQU     $-0F0
                                                  ;L0021    L0102 CC  L0112 CC  L0
116 CC  L0124 CC  L012D CC  L0136 CC  L013A CC
L0021   EQU     $-0DF
                                                  ;L002C    L0100 DI
L002C   EQU     $-0D4
                                                  ;L0030    L013F DI
L0030   EQU     $-0D0
                                                  ;L004C    L0138 DI
L004C   EQU     $-0B4
                                                  ;L0061    L0152 DI
L0061   EQU     $-9F
                                                  ;L0063    L0190 DI
L0063   EQU     $-9DH
                                                  ;L006A    L0185 DI
L006A   EQU     $-96
                                                  ;L006F    L0163 DI
L006F   EQU     $-91
                                                  ;L007A    L0156 DI
L007A   EQU     $-86
                                                  ;L0082    L016E DI  L0170 DI  L0
194 DI  L0197 DI
L0082   EQU     $-7E
                                                  ;L0083    L0173 DI
L0083   EQU     $-7DH
                                                  ;L008E    L0190 DI
L008E   EQU     $-72
                                                  ;L00A2    L016E DI
L00A2   EQU     $-5E
                                                  ;L00DF    L015A DI
L00DF   EQU     $-21
                                                  ;L0100    L019E CI
L0100:  MOV     AH,2C    ;','                     ;0100 B4 2C
        INT     21                                ;0102 CD 21
        MOV     BL,DH                             ;0104 8A DE
        AND     BL,0F                             ;0106 80 E3 0F
        MOV     CX,L015D                          ;0109 8B 0E 5D 01
                                                  ;L010D    L0126 CJ
L010D:  MOV     DX,OFFSET L015F                   ;010D BA 5F 01
        MOV     AH,9                              ;0110 B4 09
        INT     21                                ;0112 CD 21
        MOV     AH,1                              ;0114 B4 01
        INT     21                                ;0116 CD 21
        CALL    L013C                             ;0118 E8 21 00
        CMP     BL,AL                             ;011B 3A D8
        JZ      L0131                             ;011D 74 12
        MOV     DX,OFFSET L0169                   ;011F BA 69 01
        MOV     AH,9                              ;0122 B4 09
        INT     21                                ;0124 CD 21
        LOOP    L010D                             ;0126 E2 E5
        MOV     DX,OFFSET L018B                   ;0128 BA 8B 01
        MOV     AH,9                              ;012B B4 09
        INT     21                                ;012D CD 21
        JMP     SHORT   L0138                     ;012F EB 07

                                                  ;L0131    L011D CJ
L0131:  MOV     DX,OFFSET L0179                   ;0131 BA 79 01
        MOV     AH,9                              ;0134 B4 09
        INT     21                                ;0136 CD 21
                                                  ;L0138    L012F CJ
L0138:  MOV     AH,4C    ;'L'                     ;0138 B4 4C
        INT     21                                ;013A CD 21
                                                  ;L013C    L0118 CC
L013C:  CALL    L0152                             ;013C E8 13 00
        SUB     AL,30    ;'0'                     ;013F 2C 30
        JB      L014D                             ;0141 72 0A
        CMP     AL,9                              ;0143 3C 09
        JBE     L0150                             ;0145 76 09
        SUB     AL,7                              ;0147 2C 07
        CMP     AL,10                             ;0149 3C 10
        JB      L0150                             ;014B 72 03
                                                  ;L014D    L0141 CJ
L014D:  STC                                       ;014D F9
        JMP     SHORT   L0151                     ;014E EB 01

                                                  ;L0150    L0145 CJ  L014B CJ
L0150:  CLC                                       ;0150 F8
                                                  ;L0151    L014E CJ
L0151:  RET_NEAR                                  ;0151 C3

                                                  ;L0152    L013C CC
L0152:  CMP     AL,61    ;'a'                     ;0152 3C 61
        JB      L015C                             ;0154 72 06
        CMP     AL,7A    ;'z'                     ;0156 3C 7A
        JA      L015C                             ;0158 77 02
        AND     AL,0DF                            ;015A 24 DF
                                                  ;L015C    L0154 CJ  L0158 CJ
L015C:  RET_NEAR                                  ;015C C3

                                                  ;L015D    L0109 DR
L015D:  ADD     AL,0                              ;015D 04 00
                                                  ;L015F    L010D DI
L015F:  OR      AX,OFFSET L090A                   ;015F 0D 0A 09
        DEC     SI                                ;0162 4E
        DB      6F      ??                        ;0163 6F
        AND     CS:[DI],BH                        ;0164 2E 20 3D
        AND     [SI],AH                           ;0167 20 24
                                                  ;L0169    L011F DI
L0169:  POP     ES                                ;0169 07
        OR      [BX+DI],CX                        ;016A 09 09
        MOV     CL,AH                             ;016C 88 E1
        DB      82,0A2  ??                        ;016E 82 A2
        SBB     AH,82                             ;0170 82 DC 82
        MOV     BH,83                             ;0173 B7 83
        XCHG    CX,[DI]                           ;0175 87 0D
        OR      AH,[SI]                           ;0177 0A 24
                                                  ;L0179    L0131 DI
L0179:  POP     ES                                ;0179 07
        POP     ES                                ;017A 07
        POP     ES                                ;017B 07
        POP     ES                                ;017C 07
        POP     ES                                ;017D 07
        POP     ES                                ;017E 07
        POP     ES                                ;017F 07
        POP     ES                                ;0180 07
        OR      [BP+SI+L826E],AX                  ;0181 09 82 6E 82
        DB      6A      ??                        ;0185 6A
        OR      WORD PTR [BX+DI+0DH],OFFSET L240A
                                                  ;0186 81 49 0D 0A 24
                                                  ;L018B    L0128 DI
L018B:  OR      AX,OFFSET L090A                   ;018B 0D 0A 09
        OR      [BX+DI],CX                        ;018E 09 09
        DB      8E,63   ??                        ;0190 8E 63
        XCHG    SP,AX                             ;0192 94
        DEC     DI                                ;0193 4F
        ADD     CH,82                             ;0194 82 C5 82
        MOV     CH,82                             ;0197 B5 82
        MOV     BP,OFFSET L5E5E ;'^^'             ;0199 BD 5E 5E
        CMP     CX,[DI]                           ;019C 3B 0D
        OR      AH,[SI]                           ;019E 0A 24
                                                  ;L090A    L015F DI  L018B DI
L090A   EQU     $+76A
                                                  ;L240A    L0186 DI
L240A   EQU     $+226A
                                                  ;L5E5E    L0199 DI
L5E5E   EQU     $+5CBE
                                                  ;L826E    L0181 DM
L826E   EQU     $-7F32
        S0000   ENDS
;
END     L0100
```

図9　ASMGENの使用法

書式：A:¥>ASMGEN <bin-file>, [<asm-file>], [<tbl-file>]；

<bin-file>
　入力ファイル名（実行可能なCOM，EXEファイル）

[<asm-file>]
　出力ファイル名（アセンブラソースコードファイル；ASM）

[<tbl-file>]
　XREFファイル名（リファレンステーブルファイル；TBL）

図10　ASMGENの定義ファイルの書式

```
addr    command         label   ;comment
```

リスト8　KZAT_AG.SEQ

```
;addr   command label   ;comment
;-----------------------------------------------------------------
0100    /H              ; 数値には "H" をつける
0100    /T              ;ASM ファイルにオブジェクトコードをコメントとして入れる
;
0100    C       start   ;100 からはコード
010D    C       ok
0131    C       exit
0138    C       loop0
013C    C       var
014D    C       not_num
0150    C       num
0151    C       var_ret
0152    C       con_cap
015C    C       cncp_ret
015D    W       lop_cnt ;15D はワード値
015F    S       in_msg  ; 文字列
0169    S       ng_msg
0179    S       ok_msg
018B    S       end_ms
```

アセンブラ用語集
マクロ　プログラム中に繰り返し使われる命令ブロックをマクロ定義し，アセンブルするとき対応する命令ブロックに置き換える。

価格：2,000円（ドネーション）
入手先：PDS & シェアウェアライブラリ
　MS-DOS Special 1
　AGREX（株）アグレックス
〒151東京都渋谷区初台1-47-1 小田急西新宿ビル
電話：03-3377-8161

ZZ86
スクリーンエディタタイプの逆アセンブラ

これは，The BASIC誌などによく執筆されている出海勝富氏の（株）マイクロ・オーグから販売されている製品です．スクリーンエディタふうの画面からソースプログラムの作成と解析ができます．

パッケージ（ビニール袋）には5"2HDのディスクとクリップで止めただけの523ページのマニュアルが入っています．また，ディスクにはTurbo C 2.0でコンパイルできる全ソースが含まれています(図11)．しかし，このソースを変更して作ったプログラムは，その変更の程度と無関係に配布も販売もできないそうです．

インストールには，専用のバッチファイルが用意されています．

ZZ86の使用法

起動するには，まずプロジェクトファイルというものを用意しなければなりません．拡張子は".PRJ"です．これには，CPUの指定とプログラムのタイプおよびプログラム名を記入します(図12)．

用意ができたら，これを指定してZZ86を起動します．

```
A:¥>ZZ86 KZAT.PRJ
```

起動後は，プログラムを逆アセンブルしたリストを1ページ分表示し，その画面からラベルの登録などの解析作業をします．操作はおもにファンクションキーを用いて行います．

[F3] でソースリストが出力されますが，このままではアセンブルできません．KZAT.COMをZZ86で解析して得られたアセンブラソースがリスト9です．これを見ればわかる通り，セグメントのASSUME疑似命令がないのです．アセンブルするためには，たとえばCOMならファイルの先頭に図13の①，最後に図13の②を挿入しなければなりません．EXEなどでは，データのところへそれぞれ挿入する必要があります．

また，データ部分をまとめるということもしませんので，文字列などは別途テキストエディタなどで入力するほかありません．ジャンプやサーチなどの機能は豊富ですし，16進ダンプの表示など便利なところもありますが，ソースジェネレータとして用いるには問題があります．

製品名：ZZ86
　Visual Interructive Un-Assembler V1.02
　ZZ series for 8086 80286
価格：18,000円

図11　ZZ86製品ディスクの内容

```
A:¥
├INSTALL .BAT    UPDATE .DOC    ZZ86    .DOC    ZZ86    .EXE
├<SAMPLE>┐
│        ├TST286  .ASM    TST286  .BIN    DMP     .EXE    TST286  .LST
│        └TEST    .PRJ    TST286  .PRJ
├<SOC >──┐
│        ├86DMP   .C      86ERROR .C      86HELP  .C      86INIFN .C
│        ├86JUMP  .C      86LABEL .C      86MAIN  .C      86MAPSOC.C
│        ├86SEARCH.C      86WIN   .C      86WRBLD .C      86X86   .C
│        ├86XFER  .C      86      .H      COMCODE .H      ERROR   .H
│        ├LOCAL   .H      MAIN    .H      STRUCT  .H      SWITCH  .H
│        ├WIN     .H      KEYCODE .H      MAKEFILE.       ZZ86    .PRJ
│        └ZZ86    .LNK    XFERLOC .H      TYPEDEF .H
├<TBL >──┐
│        ├Z86-0   .HLP    Z86CTRL .HLP    Z86F1   .HLP    Z86F10  .HLP
│        ├Z86F2   .HLP    Z86F3   .HLP    Z86F4   .HLP    Z86F5   .HLP
│        ├Z86F6   .HLP    Z86F7   .HLP    Z86F8   .HLP    Z86F9   .HLP
│        ├Z86IELSE.HLP    Z86INT21.HLP    Z86MEMO .HLP    Z86SF10 .HLP
│        ├Z86SF5  .HLP    Z86SF6  .HLP    Z86SF7  .HLP    Z86SF8  .HLP
│        └ZZ86    .MSG    80286   .TBL    8086    .TBL
└<UTL >──┐
         └EXERELOC.C      EXERELOC.EXE
```

図12　ZZ86のプロジェクトファイルの例

```
8086
COM_NAME        KZAT.COM
```

図13　アセンブルに必要なASSUME疑似命令

```
①
code            segment
                assume  cs:code,ds:code

                ORG 100h
start:

②
code    ends
        end     start
```

連絡先：（株）マイクロ・オーグ
〒164 東京都中野区南台1-1-14 サンハイツ208
電話：03-3375-3306
FAX：03-3373-6605

リスト9　　KZAT_ZZ.ASM

```
          MOV     AH,02CH                    ;B4 2C   ..
          INT     21H                        ;CD 21   .!
          MOV     BL,DH                      ;8A DE   ..
          AND     BL,00FH                    ;80 E3 0F        ...
          MOV     CX,DS:[0015DH]             ;8B 0E 5D 01     ..].
P_0010D:MOV     DX,0015FH                    ;BA 5F 01        ...
          MOV     AH,009H                    ;B4 09   ..
          INT     21H                        ;CD 21   .!
          MOV     AH,001H                    ;B4 01   ..
          INT     21H                        ;CD 21   .!
          CALL    NEAR PTR P_0013C           ;E8 21 00        .!.
          CMP     BL,AL                      ;3A D8   :.
          JE      P_00131                    ;74 12   t.
          MOV     DX,00169H                  ;BA 69 01        .i.
          MOV     AH,009H                    ;B4 09   ..
          INT     21H                        ;CD 21   .!
          LOOP    P_0010D                    ;E2 E5   ..
          MOV     DX,0018BH                  ;BA 8B 01        ...
          MOV     AH,009H                    ;B4 09   ..
          INT     21H                        ;CD 21   .!
          JMP     SHORT P_00138              ;EB 07   ..
P_00131:MOV     DX,00179H                    ;BA 79 01        .y.
          MOV     AH,009H                    ;B4 09   ..
          INT     21H                        ;CD 21   .!
P_00138:MOV     AH,04CH                      ;B4 4C   .L
          INT     21H                        ;CD 21   .!
P_0013C:CALL    NEAR PTR P_00152             ;E8 13 00        ...
          SUB     AL,030H                    ;2C 30   ,0
          JB      P_0014D                    ;72 0A   r.
          CMP     AL,009H                    ;3C 09   <.
          JBE     P_00150                    ;76 09   v.
          SUB     AL,007H                    ;2C 07   ,.
          CMP     AL,010H                    ;3C 10   <.
          JB      P_00150                    ;72 03   r.
P_0014D:STC                                  ;F9      .
          JMP     SHORT P_00151              ;EB 01   ..
P_00150:CLC                                  ;F8      .
P_00151:RETN                                 ;C3      .
P_00152:CMP     AL,061H                      ;3C 61   <a
          JB      P_0015C                    ;72 06   r.
          CMP     AL,07AH                    ;3C 7A   <z
          JNBE    P_0015C                    ;77 02   w.
          AND     AL,0DFH                    ;24 DF   $.
P_0015C:RETN                                 ;C3      .
          ADD     AL,000H                    ;04 00   ..
          OR      AX,0090AH                  ;0D 0A 09        ...
          DEC     SI                         ;4E      N
          DB       06FH                         ;6F      o
          AND     CS:[DI],BH                 ;2E 20 3D        . =
          AND     [SI],AH                    ;20 24    $
          POP     ES                         ;07      .
          OR      [BX+DI],CX                 ;09 09   ..
          MOV     CL,AH                      ;88 E1   ..
          DB       082H                         ;82      .
          DB       0A2H                         ;A2      .
          DB       082H                         ;82      .
          DB       0DCH                         ;DC      .
          DB       082H                         ;82      .
          DB       0B7H                         ;B7      .
          DB       083H                         ;83      .
          DB       087H                         ;87      .
          OR      AX,0240AH                  ;0D 0A 24        ..$
          POP     ES                         ;07      .
          POP     ES                         ;07      .
          POP     ES                         ;07      .
          POP     ES                         ;07      .
          POP     ES                         ;07      .
          POP     ES                         ;07      .
          POP     ES                         ;07      .
          POP     ES                         ;07      .

          OR      [BP+SI+0826EH],AX          ;09 82 6E 82     ..n.
          DB       06AH                         ;6A      j
          OR      Word Ptr [BX+DI+13],0240AH ;81 49 0D 0A 24
'I..$
          OR      AX,0090AH                  ;0D 0A 09        ...
          OR      [BX+DI],CX                 ;09 09   ..
          DB       08EH                         ;8E      .
          DB       063H                         ;63      c
          DB       094H                         ;94      .
          DEC     DI                         ;4F      O
          ADD     CH,082H                    ;82 C5 82        ...
          MOV     CH,082H                    ;B5 82   ..
          MOV     BP,05E5EH                  ;BD 5E 5E        .^^
          CMP     CX,[DI]                    ;3B 0D   ;.
          OR      AH,[SI]                    ;0A 24   .$
```

MDe ◆ マルチファイルダンプエディタ

これはマシン語エディタでソースジェネレータではありませんが，ラベルつきの逆アセンブルリストを作成する機能があります．これから，アドレスとマシンコードを削除して，ASSUME疑似命令を挿入すればアセンブラソースにすることも可能です（いま，ソース出力用オプションを追加してもらうようお願いしているところです）．

それはともかく，非常に多彩な機能をもったエディタなので，プログラムの解析にはぜひ勧めたいツールです．MDeはPC-9801用ですが，IBM PS/55用にMDe55というのがDukeさんによって移植されています．

現時点でのバージョンは4.50です．図14に圧縮ファイルの中身を示します．

図14　MDEV450.LZHの内容

```
DA.INF            973     602  91-01-31 15:31:58
MDE.EXE        137966   73755  91-01-31 23:29:50
MDE.HLP         29282    8190  91-01-31 15:29:56
MDE.MAN         86385   28404  91-02-01 14:31:10
MDECUST.EXE     32108   17686  91-02-01 01:40:34
MDEDIS.INF       5499    2201  91-01-30 10:40:20
README           4102    2001  91-02-01 14:52:14
SCEDIT.EXE      30686   17542  91-01-16 15:38:06
SHORTCUT.INF     2977    1167  91-01-31 15:32:20
```

MDeの使用法

MDeの使い方を図15に示します．

ファイルは最大10個まで同時に指定できます．また，無指定で起動するとファイル選択画面になります．

オプションやコマンドは，ここではとても説明しきれないので省きますが，逆アセンブル機能は，[F3]の編集機能

アセンブラ用語集
メモリモデル　インテル社の80×86CPUなどのようにセグメントという概念から派生したメモリの管理方法

メニュー中の「/U　逆アセンブル出力」を選択することで使用できます．

KZAT.COMをMDeで解析して得られたアセンブラソースがリスト10です．割り込み命令に付いているコメントは，MDEDIS.INFを書き換えることによって，自分なりのものに変更することが可能です．

プログラム名：MDe（フリーソフトウェア）
T.Shibazaki's
Multi file Dump Editor　Ver.4.50
Author：QFG01715 T.Sibazaki
MDe55 NCA01340 Duke

入手先：Network Information Forum
NIFTY-Serve(FGALBT LIB5, FLABO LIB3)

図15　MDeの使用法

書式：MDE　［オプション］　［ファイル名…］

リスト10　　KZAT_MDE.ASM

```
;;
;;      << MDe.EXE Ver4.50  8086 Disassemble Ver 2.10 / by T.Shibazaki >>
;;
;;      Editing File Name : KZAT.COM
;;      Original File Size : 160 Byte(s)
;;      Output File Name   : KZAT.LST
;;      Output Range       : 0H to 9FH
;;      Output Bytes       : 160 Byte(s)
;;
000000 : B4 2C                 mov     ah,2CH          ; ',' Decimal=44
000002 : CD 21                 int     21H             ; (Func:2CH) Get Time
000004 : 8A DE                 mov     bl,dh
000006 : 80 E3 0F              and     bl,0FH          ; '.' Decimal=15
000009 : 8B 0E 5D 01           mov     cx,word ptr [015DH]
                      LAB_0001:
00000D : BA 5F 01              mov     dx,015FH        ; Decimal=351
000010 : B4 09                 mov     ah,09H          ; Decimal=9
000012 : CD 21                 int     21H             ; (Func:09H) Display String
000014 : B4 01                 mov     ah,01H          ; Decimal=1
000016 : CD 21                 int     21H             ; (Func:01H) Character Input With Echo
000018 : E8 21 00              call    LAB_0004        ; (00003CH)
00001B : 3A D8                 cmp     bl,al
00001D : 74 12                 jz      LAB_0002        ; (000031H)
00001F : BA 69 01              mov     dx,0169H        ; Decimal=361
000022 : B4 09                 mov     ah,09H          ; Decimal=9
000024 : CD 21                 int     21H             ; (Func:09H) Display String
000026 : E2 E5                 loop    LAB_0001        ; (00000DH)
000028 : BA 8B 01              mov     dx,018BH        ; Decimal=395
00002B : B4 09                 mov     ah,09H          ; Decimal=9
00002D : CD 21                 int     21H             ; (Func:09H) Display String
00002F : EB 07                 jmps    LAB_0003        ; (000038H)
                      LAB_0002:
000031 : BA 79 01              mov     dx,0179H        ; Decimal=377
000034 : B4 09                 mov     ah,09H          ; Decimal=9
000036 : CD 21                 int     21H             ; (Func:09H) Display String
                      LAB_0003:
000038 : B4 4C                 mov     ah,4CH          ; 'L' Decimal=76
00003A : CD 21                 int     21H             ; (Func:4CH) Terminate Process with Return C
ode
                      LAB_0004:
00003C : E8 13 00              call    LAB_0008        ; (000052H)
00003F : 2C 30                 sub     al,30H          ; '0' Decimal=48
000041 : 72 0A                 jb      LAB_0005        ; (00004DH)
000043 : 3C 09                 cmp     al,09H          ; Decimal=9
000045 : 76 09                 jbe     LAB_0006        ; (000050H)
000047 : 2C 07                 sub     al,07H          ; Decimal=7
000049 : 3C 10                 cmp     al,10H          ; Decimal=16
00004B : 72 03                 jb      LAB_0006        ; (000050H)
                      LAB_0005:
00004D : F9                    stc
00004E : EB 01                 jmps    LAB_0007        ; (000051H)
                      LAB_0006:
000050 : F8                    clc
                      LAB_0007:
000051 : C3                    ret
                      LAB_0008:
000052 : 3C 61                 cmp     al,61H          ; 'a' Decimal=97
000054 : 72 06                 jb      LAB_0009        ; (00005CH)
000056 : 3C 7A                 cmp     al,7AH          ; 'z' Decimal=122
000058 : 77 02                 ja      LAB_0009        ; (00005CH)
00005A : 24 DF                 and     al,DFH          ; Decimal=223
                      LAB_0009:
00005C : C3                    ret
00005D : 04 00                 add     al,00H          ; Decimal=0
00005F : 0D 0A 09              or      ax,090AH        ; Decimal=2314
000062 : 4E                    dec     si
000063 : 6F                    outsw
000064 : 2E                    cs:
000065 : 20 3D                 and     byte ptr [di],bh
000067 : 20 24                 and     byte ptr [si],ah
000069 : 07                    pop     es
00006A : 09 09                 or      word ptr [bx+di],cx
00006C : 88 E1                 mov     cl,ah
00006E : 82 A2 82 DC 82        and     byte ptr [bp+si-237EH],82H          ; '.' Decimal=130
000073 : B7 83                 mov     bh,83H          ; Decimal=131
000075 : 87 0D                 xchg    cx,word ptr [di]
000077 : 0A 24                 or      ah,byte ptr [si]
000079 : 07                    pop     es
00007A : 07                    pop     es
00007B : 07                    pop     es
00007C : 07                    pop     es
00007D : 07                    pop     es
00007E : 07                    pop     es
00007F : 07                    pop     es
000080 : 07                    pop     es
000081 : 09 82 6E 82           or      word ptr [bp+si-7D92H],ax
000085 : 6A 81                 push    81H
000087 : 49                    dec     cx
000088 : 0D 0A 24              or      ax,240AH        ; Decimal=9226
00008B : 0D 0A 09              or      ax,090AH        ; Decimal=2314
00008E : 09 09                 or      word ptr [bx+di],cx
000090 : 8E                    db      8EH             ; ?? Unknown Code
000091 : 63                    db      63H             ; ?? Unknown Code
000092 : 94                    xchg    ax,sp
000093 : 4F                    dec     di
000094 : 82 C5 82              add     ch,82H          ; '.' Decimal=130
000097 : B5 82                 mov     ch,82H          ; Decimal=130
000099 : BD 5E 5E              mov     bp,5E5EH        ; Decimal=24158
00009C : 3B 0D                 cmp     cx,word ptr [di]
00009E : 0A 24                 or      ah,byte ptr [si]
```

How to 逆アセンブル

SYMGENを使った逆アセンブルの実際

さて，ここでは私のSYMGENによる解析の実際を，圧縮ファイルに納められているSAMPLE.COMを例にとって解説していきます．

まず，とりあえず一度ジェネレートしてみます（図16）．

この時点で，すでに図17のようなファイルが作られています．

スタートアドレスより前とエンドアドレスよりあとのラベルは，本来，即値入力のところをオフセット値としたために作られたものと考えられます．自動的に作成される“ENT”ファイルにIサブオプションで，それらのラベルが作られたアドレスを書き出しますので，これを使ってもう一度ジェネレートすることによって，これらのラベルをほとんど消すことができます．また，ジャンプテーブルのデータもC/Jサブオプションで，自動的に書き出します．

処理を続けましょう．何かキーを押してください（図18）．

これだけでもいちおうデータ領域の分離はされていますが，残念ながらまだ完全とはいえません．これから先の解析には『MDe』や『EDIX』(拙作，NIFTY-Serve FGALST LIB6）などのマシンコードエディタが必要です．

これらで，SAMPLE.COMを覗いてみます（図19）．

どうやら，501Hから5A4Hまではデータ領域のようです．

データ領域と判断する決め手は，

アセンブラ用語集
ラベル　プログラムの中で数値や文字列，そしてアドレスを参照したときに使われる名前のこと．

図16 SYMGENでジェネレートする

```
A:¥>SGEN SAMPLE /C /D        ←この行を入力
A:¥>echo off
A:¥>symdeb SAMPLE.com < SAMPLE.cmd > SAMPLE.dis
A:¥>goto sgen
A:¥>echo off
A:¥>symgen SAMPLE.dis /C /D

SYMGEN Version 1.31 (C) 1990, 1991 By Sey-ju-row

        ただいまラベルを登録中です。    PASS1

        ＭＡＳＭのソースに変換します。  PASS2

Generated!

A:¥>echo off

新しいSAMPLE.entを作成しましたので、もう一度 SYMGEN を実行します。[中断はSTOP]

準備ができたらどれかキーを押してください
```

①文字列として読める
②00Hが多い
③アセンブルリストに脈絡がない
④メインの流れからその領域に分岐していない

などです.
以上のことから，717Hからエンドアドレスまでもデータ領域と考えられます.

図17 SYMGENの出力ファイル

```
;SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
;                       SAMPLE  .ASM
;
;               Original:       1991-01-01
;               Generate:       1991-02-18
;               Address :       0100-0937
;
;SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

.286p

code    segment
        assume  cs:code,ds:code

        ORG     000fh

b_000f  label byte

        ORG     0076h

b_0076  label byte

        ORG     0100h
start:
        jmp     j_05a5
j_0103:
        mov     ax,4c00h                ;19456
        int     21h                     ;MSDOS ｼｽﾃﾑ･ｺｰﾙ  AX=4c00
                                        ;ﾌﾟﾛｾｽの終了 in<AL=ﾘﾀｰﾝ･ｺｰﾄﾞ
c_0108:
              .
              .
j_0500:
        ret

b_0501  db      0

b_0502  label byte
        db      0

w_0503  dw      0a000h

              .
              .
j_05a5:
        push    ds
              .
              .
```

<SAMPLE.CRF>

```
c_0108  [0152][0265][04B8][05D8][061A][062C]
c_0125  [024B][05B7]
c_013A  [028C][0668]
c_015B  [0155][0186]
c_0185  [0630]
c_01F5  [05E4][061E]
c_0242  [0637][0664]
c_02BA
c_02CC  [02B3][04F1]
c_0312  [02AB]
c_033C  [032A][032E][0334]
              .
              .
              .
```

<SAMPLE.DIS>

```
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [80286]
-U 100 938
2CB2:0100 E9A204        JMP     05A5
2CB2:0103 B8004C        MOV     AX,4C00
2CB2:0106 CD21          INT     21
2CB2:0108 50            PUSH    AX
2CB2:0109 57            PUSH    DI
2CB2:010A A01605        MOV     AL,[0516]
2CB2:010D D0E0          SHL     AL,1
2CB2:010F 32E4          XOR     AH,AH
2CB2:0111 8BF8          MOV     DI,AX
2CB2:0113 8BBD2E05      MOV     DI,[DI+052E]
2CB2:0117 A01705        MOV     AL,[0517]
2CB2:011A D0E0          SHL     AL,1
2CB2:011C 03F8          ADD     DI,AX
2CB2:011E 893E0B05      MOV     [050B],DI
2CB2:0122 5F            POP     DI
2CB2:0123 58            POP     AX
2CB2:0124 C3            RET
                 .
                 .
                 .
                 .
```

<SAMPLE.ENT>

```
T S
M2
R1
P3
;
0501 054b
0595 05a4
06d3 06dd
0717 079f
;
I 0550
I 0555
I 0560
I 0671
I 0681
I 0691
I 0849
I 087e
I 0887
I 0895
I 089c
I 08a1
I 08a7
I 08ad
I 0937
;
C 069e
C 06ab
C 06b8
C 06c5
C 06d1
C 06e7
C 06eb
C 06ef
C f7f3
```

アセンブラ用語集
レジスタ　データの記憶，各種演算処理，アドレス指定などを行うためのCPU内部の記憶領域

反対に，6D3Hから6DDHのところは図20のようになっていて，データラベルがありませんからコード領域のようです．

SAMPLE.COMを動かしてみるとわかりますが，これのフローチャートは大ざっぱに図22のようになります．

SAMPLE.ASMの，各コマンドの処理のところを見てみると図23のようになっているので，B_076Cから11Hバイトがコマンドキーテーブルで，W_077Eからが各分岐アドレスであるのがわかります．

さらにわかりやすくするために，ラベル名の登録をしておくとよいでしょう（図24）．

さて，このへんで一度アセンブルしてみましょう(図25)．ただし，アセンブルする前にファイル名を変えておくのを忘れないように……．

エラー発生箇所は，IO.SYSのワークエリアをアクセスしているところで，SIの入力が即値であるべきところです．この場合はここ1か所だけですので，“.ENT”ファイルのIサブオプションを追加して再ジェネレートするまでもなく，テキストエディタでoffset b_0110のところを110hとするだけでノーエラーでアセンブラを通りますし，元のプログラムとFCで比較しても違いは出ません(^_^)

以上で，みごとに？　アセンブラのソースジェネレートに成功したわけです．

図18　SYMGENによる2度目のソースジェネレート

```
A:¥>symgen SAMPLE.dis /e /C /D < SAMPLE.ent

SYMGEN Version 1.31 (C) 1990, 1991 By Sey-ju-row

                  .
                  .
                  .
                  .

Generated!

A:¥>
```

図19　マシンコードエディタでSAMPLE.COMを覗く

```
segoffset +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F  sum 0123456789abcdef
38CD:0500 C3 00 00 00 A0 00 A2 00-A1 00 A3 00 00 1B 5B 30  :EF ﾃ.....｢..｣...[0
38CD:0510 30 3B 30 30 48 24 00 00-1A 05 00 00 00 00 00 00  :56 0;00H$..........
38CD:0520 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  :00 ................
38CD:0530 A0 00 40 01 E0 01 80 02-20 03 C0 03 60 04 00 05  :93 ..@.... .ﾀ.`...
38CD:0540 A0 05 40 06 E0 06 80 07-20 08 C0 08 60 09 00 0A  :BB ..@.... .ﾀ.`...
38CD:0550 A0 0A 40 0B E0 0B 80 0C-20 0D C0 0D 60 0E 00 0F  :E3 ..@.... .ﾀ.`...
38CD:0560 A0 0F 00 00 00 00 00 00-00 00 00 00 00 00 00 00  :AF ................
38CD:0570 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  :00 ................
38CD:0580 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  :00 ................
38CD:0590 00 00 00 00 00 18 00 00-A0 00 60 0E 00 20 40 60  :E6 ...........  @`
38CD:05A0 80 A0 C0 E0 E1 1E B8 60-00 8E D8 BE 10 01 8A 04  :9A ..ﾀ颯ｸ`.借t....
38CD:05B0 8A 64 0C 1F A3 16 05 E8-6B FB E8 B1 00 E8 BE 00  :64 蠎..｣..鏤.畦.要.
38CD:05C0 B0 00 A2 94 05 04 19 A2-95 05 E8 53 FE 50 B0 17  :94 ｰ.｢....｢..鑒.P-.
38CD:05D0 A2 16 05 B0 00 A2 17 05-E8 2D FB 58 C6 06 02 05  :66 ｢..ｰ.｢...ｰX=...
38CD:05E0 01 BE 17 07 E8 0E FC C6-06 6A 07 14 90 C6 06 6B  :E7 .ｾ.....ﾆ.j..脆.k
38CD:05F0 07 0A 90 50 A0 A4 05 24-1F 0A 06 A1 05 24 E1 0C  :44 ..娠..$....$..
-----------------------------------------------------------------
check_sum D7 3B 0A DC 99 DA 10 EE C8 4C F3 F5 89 7F 7C 45  :2E
```

図20　データラベルがない

```
                          <SAMPLE.ASM>
            .
            .
c_06d1:
        call    c_04d1
        db      '2茺 P',0a0h,0a4h,5,'$',1fh
        db      0ah,'|',5,0a2h,0a4h,5,'X',0c3h
            .
            .
```

図21　SAMPLE.ENT

```
<SAMPLE.ENT>

T S
M2
R1
P3
;
0501 05A4        ←データ領域の指定を訂正
0717 0938        ←データ領域の指定を訂正
;                 (06d3から06ddは自動的に削除
                  されます)
I 0550
I 0555
I 0560
I 0671
I 0681
     .
     .

A:¥>SGEN SAMPLE /D /E   ←この行を入力
            .
            .
```

図22　SAMPLE.COMのフローチャート

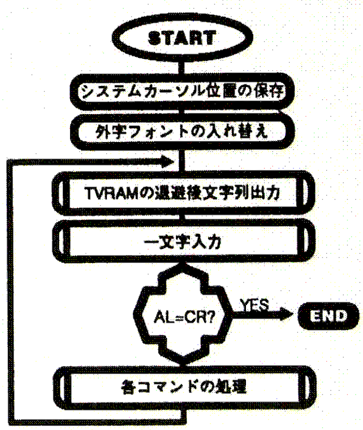


図23　各コマンド処理

```
                      <SAMPLE.ASM>
          .
          .
        mov     di,offset b_076c
        mov     cx,0011h                ;17
        repnz   scasb

        jnz     j_060a
        mov     ch,11h
        inc     cl
        sub     ch,cl
        mov     bl,ch
        xor     bh,bh
        shl     bx,1
        call    [bx+w_077e]
        jmp     short j_060a
            .
            .
b_076c  label byte

        db      1ch,1dh,1eh,1fh,'01234567VNBRU',90h

w_077e  dw      c_069e,c_06ab,c_06b8,c_06c5,c_06d1,c_06d1,c_06d1,c_06d1
        dw      c_06d1,c_06d1,c_06d1,c_06d1,c_06e7,c_06eb,c_06ef,c_06f3
        dw      c_06f7
            .
            .
```

図24　ラベルの登録

<SAMPLE.ENT>　←ユーザーラベル名の自動更新はしないので，書き換え禁止とする

```
-
T S
M2
R1
P3
;
0501 05a4
0717 0938
;
I 0671
I 0681
I 0691
;
C 069e
C 06ab
C 06b8
C 06c5
C 06d1
C 06e7
C 06eb
C 06ef
C 06f3
C 06f7
;
U 69E CUR_R
U 6AB CUR_L
U 6B8 CUR_U
U 6C5 CUR_D
U 6D1 COL_CHG
U 6E7 ATR_V
U 6EB ATR_N
U 6EF ATR_B
U 6F3 ATR_R
U 6F7 ATR_U
U 76C KEYTBL       ←ユーザーラベル名の定義
;
U 77E W JMPTBL ←ワードタイプのユーザーラベル名の定義
```

```
A:¥>SGEN SAMPLE /D /E   ←この行を入力
```

<SAMPLE.ASM>

```
        ・
        ・
keytbl  label byte

        db      1ch,1dh,1eh,1fh,'01234567VNBRU',90h

jmptbl  dw      cur_r,cur_l,cur_u,cur_d,col_chg,col_chg,col_chg,col_chg
        dw      col_chg,col_chg,col_chg,col_chg,atr_v,atr_n,atr_b,atr_r
        dw      atr_u
        ・
        ・
```

図25　アセンブルしてみる

```
A:¥>COPY SAMPLE.ASM SAMPLE0.ASM  ←この行を入力
A:¥>TASM SAMPLE0                 ←この行を入力

Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International

Assembling file:   SAMPLE0.ASM
**Error** SAMPLE0.ASM(745) Undefined symbol: B_0110
Error messages:    1
Warning messages:  None
Passes:            1
Remaining memory:  297k
```

<SAMPLE.ASM>

```
        ・
        ・
j_05a5:
        push    ds
        mov     ax,0060h                ;96
        mov     ds,ax
        mov     si,offset b_0110
        mov     al,[si]
        mov     ah,[si+0ch]
        pop     ds
        ・
        ・
```

書店で買えるCPUのマニュアル

●インテル

386™マイクロプロセッサ・ハードウェア・リファレンス・マニュアル／152ページ／5150円※
80386プログラマーズ・リファレンス・マニュアル／340ページ／6180円※
80386システム・ソフトウェア・ライターズ・ガイド／114ページ／4944円※
80387プログラマーズ・リファレンス・マニュアル／164ページ／5150円※
80286プログラマーズ・リファレンス・マニュアル／412ページ／6180円※
80286オペレーティングシステム・ライターズ・ガイド／166ページ／4635円※
80286ハードウェア・リファレンス・マニュアル／162ページ／5150円※
80286ユーティリティ・ユーザーズ・ガイド／176ページ／5150円

●モトローラ

M68000マイクロプロセッサ・ユーザーズ・マニュアル／210ページ／2580円
MC68020ユーザーズ・マニュアル／349ページ／3610円
MC68030ユーザーズ・マニュアル／631ページ／2575円
MC68030 USER'S MANUAL（英文）／676ページ／4429円※
M68000 FAMILY REFERENCE（英文）／604ページ／2060円
M68000PM／AD･･･PROGRAMMER'S REFERENCE MANUAL（英文）／576ページ／2369円

以上のマニュアルは一般の書店で注文することができる．取り扱いはCQ出版社（03-5395-2141）．
※印は次の書店に常備されている
金港堂本店（仙台市）022-225-6521／紀伊國屋書店新潟店（新潟市）0252-41-5281／八重洲ブックセンター（東京都中央区）03-3281-1811／日本橋丸善書店（東京都中央区）03-3272-7211／大盛堂書店（東京都渋谷区）03-3463-0511東大本郷生協（東京都文京区）03-3811-5407／書泉グランデ（東京都千代田区）03-3295-0011／万世書房（東京都千代田区）03-3255-0605／ラオックス ザコンピュータ館（東京都千代田区）03-5256-3111／電波堂（東京都千代田区）03-3255-8539／有隣堂（横浜市西区）045-311-6265／丸善書店（名古屋市中区）052-261-2251／紀伊國屋書店（大阪市北区）06-372-5821／コーサカテクノ（大阪市浪速区）06-543-5413／駸々堂京宝店（京都市中京区）075-223-1003／アバンティブックセンター（京都市南区）075-671-8987／淳久堂三宮店（神戸市中央区）078-392-1001／紀伊國屋書店広島店（広島市中区）082-225-3232

アセンブラ用語集
ワード　16ビット（2バイト）で表せる値をワードといい，全角コードを表すのに使われる．

処理系紹介

MS-DOS上の標準アセンブラ
MASM 5.1

MASM 5.1（Microsoft Macro Assembler Version 5.1）は，MS-DOSおよびMS OS/2を提供しているマイクロソフト社が発売している高機能マクロアセンブラである．

MASM 5.1のおもな特徴としては，その強力なマクロ機能に加え，MS-DOS，OS/2という2つのOS上で動作することがあげられる．対応しているプロセッサは，8086，80286，80386，8087，80287，80387である．

MASM 5.1のパッケージには，ソースレベルデバッガCodeView，多機能エディタMicrosoft Editorなど，さまざまなユーティリティが付属している．これらのユーティリティもMS-DOS，OS/2の両OS上で動作する．

書式上の特徴として，メモリモデルの記述が容易になる簡易化セグメントをサポートしている．また，C言語などの高級言語で作成されたオブジェクトモジュールとリンクするミクスドランゲージプログラミングにも対応している．

製品名：Microsoft Macro Assembler 5.1
価格：40,000円
問い合わせ先：マイクロソフト（株）
電話番号：03-3363-1201

Turboシリーズの秘蔵っ子
Turbo Assembler

Turbo Assemblerは，ボーランドジャパンが販売しているMASMとほぼ完全な互換性をもつ高速アセンブラである．そのアセンブルスピードは，MASMのほぼ3倍といわれている．

Turbo Assemblerのもっとも大きな特徴は，Idealモードと呼ばれる独自のアセンブルモードにある．Idealモードは，MASMの欠点といわれている部分を大幅に拡張・改良したもので，MASMの演算子やキーワードをより単純明快にし，厳密な型チェックを行うようにしている．これによって，従来ならば発見が困難だったバグをアセンブル時に発見できるようになった．

Turbo Assemblerは，"Turbo Debugger & Tools 2.0"として単独販売されているほか，Turbo C++，Turbo PascalのProfessional版にも同梱されている．これらのパッケージには，ソースレベルデバッガTurbo Debuggerやプログラムのボトルネックを取り除くTurbo Profilerなどのユーティリティが含まれている．

Turbo Assemblerが対応しているプロセッサは，8086，80186，80286，80386，80486，8087，80287，80387である．

製品名：Turbo Debugger & Tools 2.0
価格：39,800円
問い合わせ先：（株）ボーランドジャパン
電話番号：03-3486-1400

超高速な
オプティマイズアセンブラ
OPTASM

OPTASMは，米国SLR Systems社が開発したMASM 5.0互換の高速アセンブラである．1分間に75000行（MASMの約5～10倍）といわれる超高速アセンブルに加え，高度な最適化をも実現している．

ラージモデルのアプリケーションを作成する場合には，同じセグメントにあるサブルーチンをファーコールで呼び出す必要があるが，OPTASMはファーコールをPUSHとCSとニアコールで置き換えるため，高速にコードを実行することができる．また，8086系のCPU特有のセグメントの範囲外への条件ジャンプにおいても，OPTASMはこれらの処理を自動的に行い，プログラマの負担を軽くしてくれる．

OPTASMの関連商品として，高速リンカOPTLINK，高性能ライブラリアンOPTLIBなど，いくつかのユーティリティが発売されている．

OPTASMおよび関連商品の国内販売は，サザンパシフィックが行っている．

製品名：SLR OPTASM
価格：19,800円
問い合わせ先：（株）サザンパシフィック
電話番号：045-314-9514

アセンブラ学習用
オンメモリアセンブラ
RUN/MASM

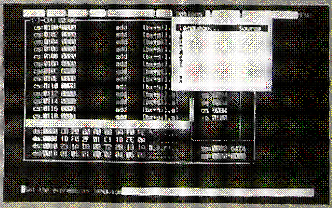
Advanced RUNシリーズは，初心者のプログラミング言語入門用に作られたシリーズで，BASICライクなユーザーインターフェース，ビルトインエディタ，デバッガなどを統合した環境を備えている．

Advanced RUN/MASMはこのAdvanced RUNシリーズのひとつで，8086/80286のアセンブラプログラムを学習することができる．RUN/MASMでは，SEGMENTやASSUMEなど，MASMの複雑な疑似命令を使用することなくアセンブラプログラミングを記述することが可能で，さらにリンク操作なしにCOMファイルを作成できるため，初心者のアセンブラ学習に最適なシステムとなっている．

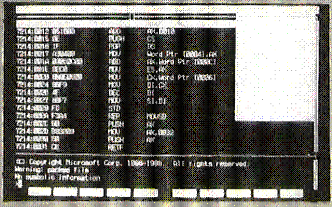
Advanced RUN/MASMのニーモニックや疑似命令は，マイクロソフト社のMASMに準拠している．

製品名：Advanced RUN/MASM
価格：29,800円
問い合わせ先：（株）ライフホート
電話番号：03-3293-4711

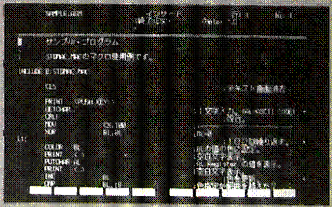
Turbo Assembler



MASM



RUN/MASM



パソコン博物館 V

日本におけるパソコンの萌芽

TK-80

日本電気 TK-80

井戸禎光

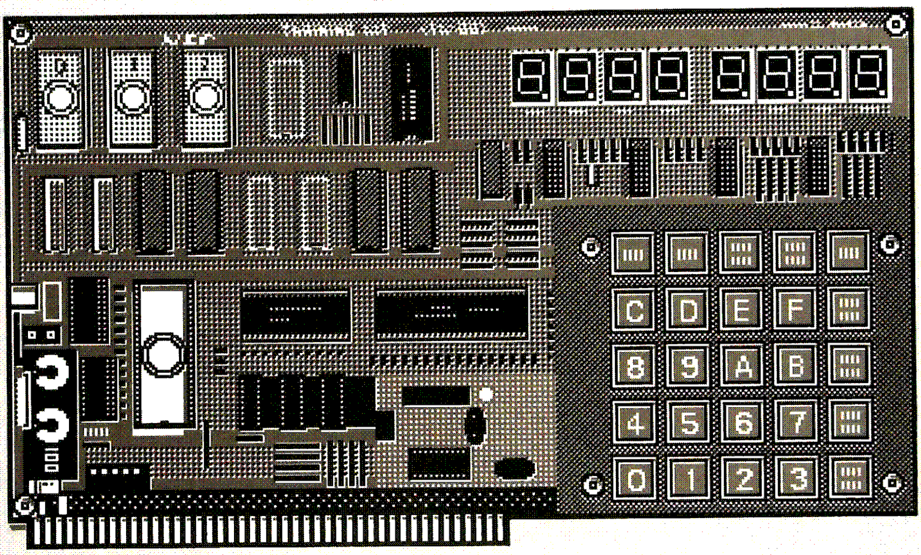


Illustration
TOSHIHITO ITO

米国内のパソコンについて第1期，第2期をひと通り見たところで，日本について状況を振り返ってみよう．米国に数年遅れて，わが国でもコンピュータの組立キットが売り出された．その最初のものが日本電気製のTK-80である．発売は1976年，8万8500円で，2年間で6万台販売した．

米国では，1974年にALTAIR8800が売り出され，1975年にBASICがのって約1万台売れたころである．1976年になると，ザイログ社のZ80が発売されている．米国における第2期のAPPLE-II，コモドールペット2001，タンディTRS-80は，いずれもまだ発売していなかったころである．

日本電気の中に電卓を主要顧客とするMOSICを製造している部門があり，そこにマイクロコンピュータの開発を行っていた部隊があった．このグループは，インテル社が4004を発表した次の年，すなわち1972年にはすでにμCOM700シリーズを開発し，1973年になると4ビットのμCOM4を発表した．続いて1974年には，16ビットのμCOM16の製品化に成功している．

このような背景があったわけではあるが，日本の社会環境のほうはマイクロコンピュータを使うようにはなっておらず，できたマイクロコンピュータもなかなか売れ行きが上がってこなかった．そこで日本電気マイコン販売部ではICの販売促進のためマイコンキットの販売を思いたち，TK-80を作った．ところが日本電気は長いことNTT（当時は日本電信電話公社）依存の体質があり，民生品を小売りするようなことは苦手のほうであった．そこで1976年にBit-INNというサービスセンターも同時に設立している．

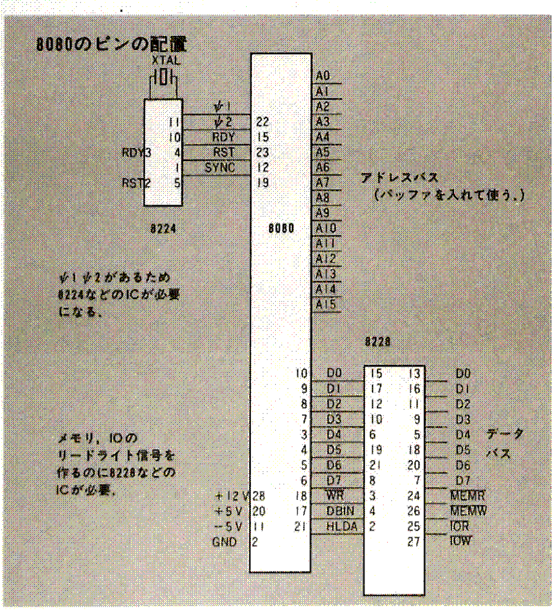
さて，事業をはじめるにあたって想定顧客を設定することがよくある．当時の日本電気においても，工場内

の設計者・技術者を想定していたというが，やはり当てがはずれ，そのかわりに，コモドール社でもそうであったように，学生たちとマニアが飛びついてきた．当初の予定と異なったのは顧客層だけでなく販売数量についても，月に数十台程度と考えていたものが五百台も売れていった．

TK-80はCPUに8080A，キーボードは16進，表示装置も16進8桁，プログラミングは機械語，記憶用のカセットテープがオプションとして取り付け可能といった性能のものであった．続いてBASIC搭載のTK-80BS（12万8000円）が売り出された．BASIC搭載のCOMPO-BS/80（20万円前後）は，箱に入った完成品であり，「キットですら5万台以上も売れたのだから完成品はもっと多く売れる」と考えられていたにもかかわらず，実際はあまり売れず，もとのTK-80が売れ続けたのは，APPLE-IIの場合と似ていて興味深い（10万円以下のものは課長・係長のサインで買えた）．

このプロジェクトを実現させた人は，トップに半導体の権威である取締役大内淳義氏，販売事業部長に沢登氏，実際のプロジェクトマネージャに渡部和也氏，メンバーとしては後藤富雄氏という豪華な布陣であった．社外には，Bit-INNから販売網の構築に大きな貢献をした日本電子販売の野口重次社長がいる（当時の役職）．

新しい商品を売り出す場合，何か古い商品の販売ルートを見つけ，それに教育を行い，代行する場合があるが，日本電気のパソコンの場合は，新たに販売ルートを作っていったのである．実際，秋葉原のBit-INN 1号店からはじまって大阪・名古屋と続き，一方，マイコンショップのほうも博多にある古村エルコンという店などからはじまった．その過程を見ると，拡充していく苦労の連続があったことは想像にあまりある．社員も休日になると秋葉原の店に立ち寄って，多くのアマチュアパソコンホビイストの質問に答えるような日が続いたとのことである．



当時，パソコンは直接，衣食住の日常生活に役立ったわけではないので，購入動機はあくまで趣味の段階であった．今でもパソコンを使おうとするとフォートランとかアセンブラとかよけいなものを勉強しおわるまで，使用はお預けになったり，ちょっとした手引書の見落としに気が付かず，何日もおかしいおかしいと思うばかりの日が続いて，そのうちにほかのことで忙しくなり，パソコンを買ったときの熱意は消えてしまい，パソコンのほうはほこりを被ってしまうことがよくある．その点で当時のNEC社は結果として的を得た対応をしていたといえよう．さずがに最近ではパッケージソフト（ゲーム・ワープロソフト・表計算など）が売り出されており，これを買って動かすことで，役にたっているケースが多い．

結局のところ，他社が実力を発揮できないままに時間がどんどんたっていったのに対し，日本電気が一人，成功への道を歩みはじめた最大の原因は大きく二つに分けられるのではないか．その一つは“技術的な面でオープンであったこと（これは米国でも同様であり，当初のIBM社のPCの成功戦略に通じる）”であり，その例をあげれば通常は契約を行っている企業にしかオープンにしない書類，たとえば回路図・モニターの中身・各種説明書・8080・メモリの規格表・サブルーチン集などをTK-80の購入者に渡したことが評判をよくした原因である．

もう一つが“組織内にうまい人の組み合わせができた”ことにあろう——すなわち“世の中レベルで見て正しい決断が素早く行えるトップ”と，夢中になって仕事をするグループリーダーとメンバー，それにプロジェクトを助ける外部の人・組織”の組み合わせなのであろう．

プログラマの道具学

第5回　猫

祐安重夫



Illustration
Mikio Sasaki

猿にタイプライタをたたかせていると，いつの日か偶然に大傑作がうちだされるという話がある．こうしてキーボードにむかって原稿をかいていると，べつに大傑作でなくてもいいから，かわりに文章をタイプしてくれないかとおもうのだが，残念ながらうちには猿はいない．

猫だったら机の横にあるUnixワークステーションの上で昼寝をしているのだが，猫の手もかりたいようないそがしいさなかでも，平然としてねむりつづけているだけである．もっとも猫はキーボードをたたいたりしないので，原稿をかいてくれたりプログラムをつくってくれるわけはない．結局，暖房より強力に部屋をあたためているワークステーションの上で，気持よさそうにねむっている猫を横目でみながら，自分でキーボードをたたきつづけるしかない．

タイプライタをたたく猿なら，簡単なプログラムによってコンピュータでシミュレーションが可能である．乱数をつかってでたらめに文字を出力すればいいだけのことだ．これではあまりに単純すぎるというのなら，辞書をもつことで意味のある単語だけを出力するようにもできる．さらに構文規則をもてば，文法的にただしい（ようにみえる）文だけを生成することも可能である．

しかしこのプログラムから，本当に意味のあるまとまった文章がうまれてくることがあるのだろうか．うまれるとしても，それは一体どれくらい先の話なのか．いや，そもそも膨大なプリントアウトをみながら，意味のある文章をみつける仕事は，誰がするのだろうか．人工知能――まさか．

ふと気がついたのだが，要するに猿のシミュレーションとは，人工知能の研究の中でおこなわれていた文章の自動生成の実験と，ほとんどかわらないではないか．



それなら猫のシミュレーションの方が，はるかにおもしろそうだ．だが，一体どうやって――．

『猫文学大全』（大和書房）と『猫百話』（ちくま文庫）という2冊の本の編者であり，みずからを「半猫人」とよぶ英文学者の柳瀬尚紀氏は，1台のパーソナル・コンピュータを「猫見習い」とよび，立派な「猫ンピュータ」とすべくそだてあげているそうだ．このコンピュータには，英語でかかれた猫文学の断片が膨大なデータベースとしておさめられ，プログラムによってそこから単語や文をひろってくることで，あたらしい猫文学の創作に日夜いそしんでいるという．

あのダグラス・R・ホフスタッターの『ゲーデル，エッシャー，バッハ』（白揚社）の訳者の一人がそだてあげただけに，この「猫見習い」の創作する猫文学は，ただものではない．

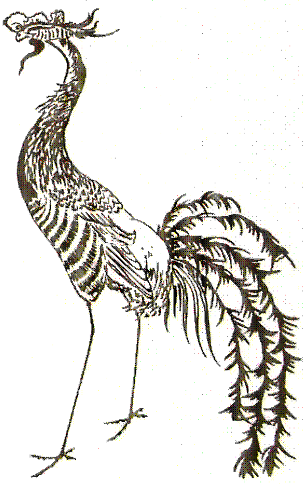
猿と猫がちがうところは，猿の出力する文章が単なる猿真似にすぎないのにたいして，猫は徹頭徹尾自分のことだけをいいつづけることだろうか．また猿の出力する文のおおくは無意味だが，猫文学はいかようにもこじつけがきくようだ．なるほど人工知能とは，要するに猿真似をするコンピュータのことだったのかと，納得した次第である．

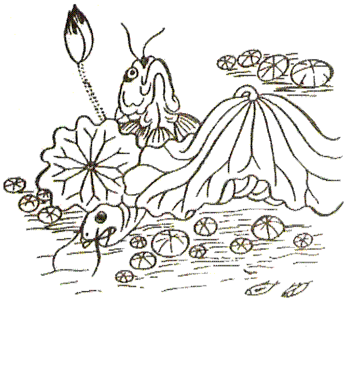
ところでうちの猫だが，さきほど目をさまし，たいしておもしろくもなさそうに机の上のマウスにじゃれたあと，おもむろに散歩にでかけていった．

プログラム遊工房

谷口典生

伍乃巻
『minesweeper』ゲーム





今回は，時事ネタにしてはちょっと古い(？)のですが，「minesweeper」……地雷を撤去するゲームです．もとはWindows用のゲームなので，できればマウス操作にしたかったところですが，キーボード操作でも十分楽しめると思います．
ワンポイントテクニックでは，再帰的呼び出しを使った関数で書き直したプログラム例と，その注意点について説明します．

リスト1

```
 1: /*
 2:  * msweeper.c             「mine sweeper」ゲーム
 3:  */
 4:
 5: #include <time.h>
 6: #include <stdlib.h>
 7:
 8: #ifdef __TURBOC__
 9: /*
10:  * Turbo C 用グラフィック初期化/復帰ルーチン
11:  */
12: #include <graphics.h>
13: void graphinit()                         /* グラフィックモード設定 */
14: {
15:         int graphdriver, graphmode;
16:
17:         graphdriver = DETECT;            /* 自動検出 */
18:         graphmode = PC98C8;              /* 8色モード */
19:         initgraph(&graphdriver, &graphmode, "");
20: }
21: void graphdeinit()                       /* グラフィックモード復帰 */
22: {
23:         restorecrtmode();                /* 画面モードを元に戻す */
24:         printf("¥x1b*");                 /* 画面クリア */
25: }
26: #else                            /* MS-C or Quick C */
27: /*
28:  * MS-C, Quick C 用グラフィック初期化/復帰ルーチン
29:  */
30: #include <graph.h>
31: #define BLACK                    0
32: #define BLUE                     1
33: #define RED                      4
34: #define MAGENTA                  5
35: #define WHITE                    7
36: #define setcolor                 _setcolor
37: #define floodfill                _floodfill
38: #define rectangle(l,t,r,b)       _rectangle(_GBORDER,l,t,r,b)
39: #define setfillstyle(a,b)        _setcolor(b)
40: void graphinit()                         /* グラフィックモード設定 */
41: {
42:         _setvideomode(_98RESSCOLOR);     /* グラフィック8色+テキストモード */
43:         printf("¥x1b[>5h");              /* カーソル消去 */
44: }
45: void graphdeinit()                       /* グラフィックモード復帰 */
46: {
47:         _setvideomode(_DEFAULTMODE);     /* デフォルト(テキスト80行)モード */
48:         printf("¥x1b[>5l¥x1b[>1l");      /* カーソル表示、最下行復帰 */
49: }
50: #endif
51:
52: #define S_NORMAL        0                /* 開かれる前 */
53: #define S_SELECT        1                /* 選択 */
54: #define S_OPEN          2                /* 開く */
55: #define S_MARK          3                /* マーキング */
56: #define M_MINE          -1               /* 地雷 */
57: #define M_OPENED        -2               /* 開かれたブロック */
58: #define M_MAGIC         9                /* マジックナンバー */
59: #define gotoxy(x,y)     printf("¥x1b[%d;%dH",y,x)
60:
61: int level;                               /* ゲームレベル */
62: int openblocks;                          /* 開いたブロックの数 */
63: int map[30][16];                         /* マップ */
64: int mark[30][16];                        /* マーキング用配列 */
65: int xsize[3] = { 8, 16, 30 };            /* 各レベルの横サイズ */
66: int ysize[3] = { 8, 16, 16 };            /* 各レベルの縦サイズ */
67: int mines[3] = { 10, 40, 99 };           /* 各レベルの地雷数 */
68:
69: void showblock(int x, int y, int mode)  /* ブロックの表示 */
70: {
71:         int bs;                                  /* ブロックのサイズ */
72:
73:         setcolor(mode==S_SELECT ? RED : WHITE);
74:
75:         bs = level==0 ? 32 : 16;                 /* レベル0のときは大きくする */
76:         if(level==0) {
77:                 rectangle(bs*x+12, bs*y+8, bs*(x+1)+11, bs*(y+1)+7);
78:                 gotoxy((x+1)*4, (y+1)*2);
79:         } else {
80:                 rectangle(bs*x+4, bs*y, bs*(x+1)+3, bs*(y+1)-1);
81:                 gotoxy((x+1)*2, y+1);
82:         }
83:         if(mode==S_OPEN) {                               /* 開ける */
84:                 if(map[x][y]>M_MAGIC)
85:                         printf("%d",map[x][y]-M_MAGIC);
86:                 setfillstyle(SOLID_FILL, map[x][y]==M_MINE?MAGENTA:BLUE);
87:                 floodfill(bs*x+13,bs*y+9, WHITE);
88:                 map[x][y] = M_OPENED;
89:                 openblocks++;
90:         }
```

プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C，Quick CおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

今回のプログラムではグラフィック関数を使っているので，MS-CやQuick Cでコンパイルするときにはグラフィックライブラリを指定します．MS-Cなら「CL msweeper.c /link graphics.lib」などとします．Turbo Cでは，実行時に.BGIファイルが必要となるので注意してください．

起動するときの書式は，

MSWEEPER {1|2|3}

と，引数として1から3の数字を指定します．この数字はゲームレベルで，レベル1では8×8のフィールドで地雷が10個，レベル2で16×16の40個，レベル3では30×16の99個と，レベルが上がるにつれて大きいフィールドになり，地雷の数も増えていきます．

それぞれのレベルのフィールドの大きさと地雷の数はプログラムの65-67行で定義されているので，ほかの組み合わせも試してみてください．

ゲームの難易度はフィールドの広さとブロック数に対する地雷の数で決まりますが，あまり地雷の割合を大きくすると運がよくないとクリアできないゲームになってしまうので，地雷の割合は20%前

```
        if(mode==S_MARK && map[x][y]!=M_OPENED) {       /* マーキング */
                mark[x][y] = !mark[x][y];
                setfillstyle(SOLID_FILL, mark[x][y]==0?BLACK:MAGENTA);
                floodfill(bs*x+13,bs*y+9, WHITE);
        }
}

void openblock(int sx, int sy)          /* 選択されたブロックを開ける */
{
        int x,y,tx,ty,loop;

        map[sx][sy]+=M_MAGIC;                   /* マジックナンバーを加算 */
        if(map[sx][sy]!=M_MAGIC) {              /* まわりに1つでも爆弾がある */
                showblock(sx, sy, S_OPEN);              /* 表示 */
                return;
        }

        do {            /* まわりに爆弾がなければまわりのブロックも開く */
            loop = 0;
            for(x=0;x<xsize[level];x++)
                for(y=0;y<ysize[level];y++) {
                    if(map[x][y]==M_MAGIC) {
                        for(tx=max(x-1,0);tx<=min(x+1,xsize[level]-1);tx++)
                            for(ty=max(y-1,0);ty<=min(y+1,ysize[level]-1);ty++)
                                if(map[tx][ty]<M_MAGIC && map[tx][ty]>=0) {
                                    map[tx][ty]+=M_MAGIC;
                                    loop = 1;
                                }
                    }
                    if(map[x][y]>=M_MAGIC)
                        showblock(x, y, S_OPEN);
                }
        } while(loop);
}

int select(int *sx, int *sy)            /* 入力 */
{
        int key;

        do {
                showblock(*sx, *sy, S_SELECT);          /* select */
                key = getch();
                showblock(*sx, *sy, S_NORMAL);          /* release */
                switch(toupper(key)) {
                case '4':  case 'H':                    /* 左移動 */
                        *sx = *sx>0 ? *sx-1 : xsize[level]-1;
                        break;
                case '6':  case 'L':                    /* 右移動 */
                        *sx = *sx<xsize[level]-1 ? *sx+1 : 0;
                        break;
                case '8':  case 'K':                    /* 上移動 */
                        *sy = *sy>0 ? *sy-1 : ysize[level]-1;
                        break;
                case '2':  case 'J':                    /* 下移動 */
                        *sy = *sy<ysize[level]-1 ? *sy+1 : 0;
                        break;
                case '5':  case 'N':                    /* マーキング */
                        showblock(*sx, *sy, S_MARK);
                        break;
                case '0':  case 'X':                    /* ････ */
                        if(openblocks==0 && map[*sx][*sy]==0)
                                openblock(*sx,*sy);
                        break;
                }
        } while(key!=' ' || map[*sx][*sy]==M_OPENED);   /* スペースで決定 */

        if(map[*sx][*sy]==M_MINE)
                return(1);                      /* 地雷に当った */

        openblock(*sx,*sy);                     /* 指定されたブロックを開ける */
        if(openblocks+mines[level]==xsize[level]*ysize[level])
                return(2);                      /* 地雷以外のブロックをすべて開けた */

        return(0);
}

void initmap()          /* マップデータを作成し、初期画面を表示する */
{
        int i,x,y,mx,my;

        openblocks = 0;
        for(x=0; x<xsize[level]; x++)
                for(y=0; y<ysize[level]; y++) {
                        map[x][y] = 0;
                        mark[x][y] = 0;
                        showblock(x,y,S_NORMAL);        /* 初期画面表示 */
                }

        for(i=0; i<mines[level]; i++) {         /* 地雷を敷設する */
                do {
                        mx = rand() % xsize[level];
                        my = rand() % ysize[level];
                } while(map[mx][my]==M_MINE);
                for(x=max(mx-1,0); x<=min(mx+1,xsize[level]-1); x++)
                        for(y=max(my-1,0); y<=min(my+1,ysize[level]-1); y++)
                                if(map[x][y]!=M_MINE)
```

後にしておくのがよいと思います.

プログラムの遊び方

このゲームの目的は，フィールド上にランダムに埋められた地雷の場所をできるだけ早くすべて捜し当てることです．

プレイヤーはブロックを自由に選んで開けることができます．もし開けた場所に地雷が埋まっていればゲームオーバーです．うまく地雷のないブロックを開くと，そのブロックのまわり8か所にいくつ地雷が埋まっているかが表示されます．もし，まわりに地雷の埋まっているブロックがないときには，まわりの8つのブロックはすべて開けても安全ということになりますが，この作業は自動的に行われます．

地雷のある場所以外のすべてのブロックを開ければゲームクリアで，かかった時間（秒数）が表示されます．

選択するブロックを上下左右に移動するには，それぞれ8，2，4，6(またはK，J，H，L)のキーを使用します．スペースを押せば，選択されている(ワクが赤く表示されている）ブロックが開きます．

5（またはN）を押すと，選択されたブロック（まだ開いていない）が紫色に塗りつぶされます（マーキング)．再度押せば元に戻ります．これは地雷が埋まっていると思われる場所に目印をつけるためのもので，ゲームのクリア条件には関係ありません．

ゲームの開始時は適当な場所を開け

```
187:                                         map[x][y]++;
188:                 map[mx][my] = M_MINE;
189:         }
190: }
191:
192: int main(int ac, char *av[])
193: {
194:         int x=0, y=0, result;
195:         time_t seedtime, starttime, endtime;
196:
197:         time(&seedtime);
198:         srand((unsigned int)seedtime);          /* ランダマイズ処理 */
199:
200:         level=atoi(av[1])-1;                    /* 引数(レベル数)を取得 */
201:         if(ac!=2 || level<0 || level>2) {
202:                 printf("usage : msweeper {1|2|3}¥n",av[0]);
203:                 exit(0);
204:         }
205:
206:         graphinit();                            /* 画面モードの初期化 */
207:         initmap();                              /* マップデータ初期化 */
208:
209:         time(&starttime);
210:         while((result=select(&x, &y))==0);      /* メインループ */
211:         time(&endtime);
212:
213:         if(result==1) {                         /* 地雷を掘り当ててしまった */
214:                 printf("¥x1b[18;1HBOMB!¥n¥a¥a¥a¥a¥a¥a");
215:         } else {                                /* すべての地雷を探し当てた */
216:                 printf("¥x1b[18;1HCONGRATULATIONS!!¥n");
217:                 printf("TIME = %d¥n", endtime-starttime);
218:         }
219:
220:         for(x=0;x<xsize[level];x++)             /* すべて開ける */
221:                 for(y=0;y<ysize[level];y++)
222:                         showblock(x,y,S_OPEN);
223:         getch();
224:         graphdeinit();                          /* 画面モード復帰 */
225: }
226:
227:
```

るしかないので，運が悪いといきなり地雷を掘り当ててしまうことになります．そういった事故死に納得できない人のために，まだ1か所も開けていないときにかぎり，0（またはX）のキーを押すと，まわりに地雷のないブロックのみ開くという機能をつけておきました．ただし，この機能に頼ると最初に大きく開いたときの感動が薄れるので，連続していきなり地雷に当たっていらいらしたときなどの使用に限るのがよいと思います．

最初のうちは，どこから開けていけばよいのか迷うと思いますが，マーキングを利用してどこに地雷があるかを予想するようにすれば，安全に開けられる場所のパターンがわかってくると思います．慣れてきてクリアできるようになったら，できるだけ短い時間でのクリアにチャレンジしてみましょう．レベル2で2分を切れるようになれば上級者です．

プログラムの説明

9-25行

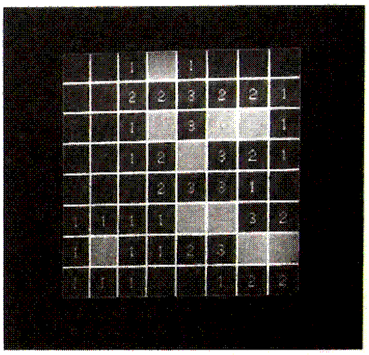
Turbo C用のグラフィック初期化ルーチン．

27-49行

MS-CおよびQuick C用のグラフィック初期化ルーチン．色（パレット）定数とグラフィック関数をTurbo Cにあわせて定義しなおしています．fill系の関数で使うパレットはTurbo Cではsetfillstyle()で設定するので，_setcolor()に置き換えて定義しています．

52-58行

各種定数を定義します．



52-55行

showblock()の引数用の定数．

56-58行

マップデータ用の定数．

59行

表示位置をy行xカラムに移動させる関数gotoxy(x, y)の定義．

65-67行

ゲームレベルに応じたフィールドの大きさと地雷の数を配列データとして定義しています．最大サイズを変更するときは，63-64行のmapとmarkの配列の大きさも変更するのを忘れないようにしてください．

❖showblock()

指定されたブロックを画面に表示します．

73行

第3引数(mode)がS_SELECTならワクの色を赤にします．

75-82行

ワクの表示．レベル0（フィールドの大きさが8×8）のときはブロックを大きく表示します．

83-90行

modeがS_OPENならマップデータ（まわりにいくつ地雷があるか）を表示し，ブロックを青で塗りつぶします．

91-95行

modeがS_MARKで，まだ開いていないブロックならブロックを紫色に塗り

つぶします．すでにマーキングされている場合は黒に戻します．

openblock()

指定されたブロックを「開ける」処理をします．

102-106行

まず，マップデータにマジックナンバー（M_MAGICで定義されているが，初期マップデータの最大値8より大きい数であればよい）を加算し，まわりに地雷のあるブロック（または地雷のある場所）ならそのまま表示して戻ります．

108-124行

まわりに地雷がない場合は，自動的にまわりのブロックをすべて開く処理を行います．開かれたブロックがさらに「まわりに地雷がない」ブロックだったら，続けてそのまわりも開く‥‥という処理を繰り返します．

select()

130-155行

開いていないブロック上で，スペースが押されるまでループします．

135-137行

4またはHが押されたら，選択されているブロックを左に移動します．左端を越えたら右端に移動します．

138-146行

同様にして，右，上，下へ選択されているブロックを移動します．

147-149行

5またはNが押されたら，マーキングを行います．マークされているブロックならマークを解除します．

150-153行

0またはXが押されれ，まだどこも開けていない状態で，さらに選択されているブロックのまわりに地雷がないときにはそのブロックを開きます．

157-158行

地雷の上でスペースを押してしまった場合には1を返します．

161-162行

地雷に当たることなく，ほかのすべてのブロックを開くことができたら2を返します．

164行

それ以外はゲーム続行で0を返します．

initmap()

171-177行

マップデータをクリアし，初期画面を表示します．

179-189行

地雷を設置します．まだ地雷の置かれていない場所をランダムに選び，まわりの8つのブロックのマップデータを加算します．for文の中でmax()，min()を使ってマップデータの配列からはみ出さないようにしていることに注意してください．

main()

197-198行

time()を利用して乱数の初期化を行っています．

200-204行

プログラムの引数がないか不正な値であれば，簡単な起動方法を表示してプログラムを終了します．

209-211行

select()からの戻り値が0以外になるまでループします．メインループの前後でtime()を使ってゲーム時間を測ります．

213-218行

地雷を掘り当ててしまった場合は「BOMB!」というメッセージとともにベルを鳴らします．うまく地雷以外のブロックをすべて開けることができたなら，「CONGRATULATIONS!」のメッセージとかかった時間を表示します．

220-222行

残ったブロックをすべて開けて，地雷の場所を表示します．

ワンポイントテクニック

再帰的関数を使う

今回のプログラムでは，関数openblock()がわかりづらいと感じた人が多いと思います．

それほど入り組んだ処理をしているわけではないのですが，インデントも深く，見通しが悪くなっています．

そもそも，この関数は何をしているのでしょうか？　ゲームの遊び方のところで書いたように，「まわりに地雷が1つもないブロック」（以下「0ブロック」と呼びます）を見つけたら，まわりの8つのブロックはすべて開けてよいことになります．その8つのブロックの中に0ブロックがあれば，さらにそのまわりの8つ（そのうちのいくつかはすでに開けられているはずですが……）のブロックも開けることができます．この処理を続けられるだけ続けるのがopenblock()の役割です．

もう一度リスト1のopenblock()を見てください．まず，指定されたブロックにマジックナンバーを加えて印をつけます．0ブロックでなければそれを表示して戻ります．0ブロックならdo～whileのループに入り，フィールド上の全ブロックを調べます．印のついた0ブロック

があれば，自分およびまわりの8つのブロック(まだ開いていないもの)に印をつけ，ループフラグを立てます．印のついた0ブロックでないブロックがあれば，そのブロックを表示します．この繰り返しによって連鎖的にブロックを開くことができるわけです．

つまり，まわりのブロックを直接開くことができない（0ブロックが含まれている可能性がある)ため，印をつけておいて，あとで全フィールドを走査するといういかにも苦しい方法をとっているのです．

この方法が複雑なプログラムになった理由は，「0ブロックのまわりのブロック」を「開く」ところまでめんどうをみているからです．しかし，openblock()自身が指定されたブロックを開く関数なのですから，これを利用しない手はありません．

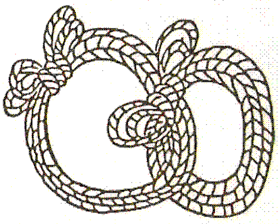
リスト2を見てください．これは0ブロックのまわりのブロックを開くためにopenblock () 自身を呼び出すようにして書き直したものです．リスト1と比べるとあっけないくらい簡単になっています．リスト1のopenblock()と入れ換えるには，すでにマジックナンバーが不要になっているので，58行目を「#define M_MAGIC 0」とするか，84-85行目をリスト3のようにと変更しなければいけません．

このように，自分自身を(直接または間接的に)呼び出す関数を「再帰的関数」といいます．再帰的関数を用いると，簡潔でわかりやすいプログラムを書くことができますが，注意すべき点もあります．

リスト4は，やはり再帰的に書いたopenblock () の例ですが，このプログラムはうまく動きません．0ブロックを表示する前に再帰的な呼び出しを行っているため，無限ループになる恐れがあります．スタックチェックを入れていれば（とくに指定しないかぎり入るはずです)，ランタイムエラー（スタックオーバーフロー）で止まるでしょう．再帰的な呼びだしによってプログラムがどのように動作するかをイメージしておかないと，このようなミスを犯しがちです．

さらに注意が必要なのは，再帰的な関数呼び出しの繰り返しによってスタックがたりなくなる可能性です．MS-CやQuick Cではデフォルトのスタックサイズが2KBしかないので，リスト3のopenblock () を使った場合，レベル3などではスタックオーバーフローがおこることがあります．爆弾の数を少なくすると，ひんぱんにおこるでしょう．解決方法としては，スタックサイズ(リンカオプションで指定する）を大きくすることが考えられます．MS-Cなら図1のようにすれば，スタックサイズは8000バイトになります．

今回のプログラムに関してはスタックを広げるだけで十分ですが，関数内で使用するローカル変数の領域や，再帰的な呼び出しが最大どのくらいの深さになるかによって必要なスタックの大きさは変わります．再帰の深さが計算できないようなときには，再帰呼び出しを使わないようなアルゴリズムにしてしまったほうが安全です．



リスト2　再帰的に書いた例

```
void openblock(int sx, int sy)
{
        int x,y;

        if(map[sx][sy]>0)
                showblock(sx, sy, S_OPEN);
        else if(map[sx][sy]==0) {
                showblock(sx, sy, S_OPEN);
                for(x=max(sx-1,0); x<=min(sx+1,xsize[level]-1); x++)
                        for(y=max(sy-1,0); y<=min(sy+1,ysize[level]-1); y++)
                                openblock(x, y);
        }
}
```

リスト3　修正部分

```
        if(map[x][y]>0)
                printf("%d",map[x][y]);
```

図1　スタックを広げる

```
CL msweeper.c graphics.lib /link /STACK:8000
```

リスト4　まちがった例

```
void openblock(int sx, int sy)
{
        int x,y;

        if(map[sx][sy]==0) {
                for(x=max(sx-1,0); x<=min(sx+1,xsize[level]-1); x++)
                        for(y=max(sy-1,0); y<=min(sy+1,ysize[level]-1); y++)
                                openblock(x, y);
        }
        if(map[sx][sy]>=0)
                showblock(sx, sy, S_OPEN);
}
```

MS-DOS TOOL LABO

MS-DOSツール・ラボ

第2回 割り込みベクタ表示・設定プログラム VECT.EXE

加野島英渡

コンピュータの世界では割り込みという言葉をよく聞くが，あんがい中身を知らないで話をしていることが多い．「ここは割り込みで処理してますので」「あー，割り込みですか」といいながら（割り込みってなんだろう?）と内心首をかしげていたりする．今月は，ちょっとこの割り込みのことを調べてみよう．

割り込みとは

人間の世界でも，割り込みというのはおなじみの言葉だ．仕事に熱中しているときに，「あー，キミ，すまんがこれを今日中にまとめてくれんか」とやられてクサった経験をおもちの方も多いだろう．このように，なにかの仕事をしているときに（あるいはなにもせずにボンヤリしていたとしても），突然まったく別の仕事が入ってくる状態が割り込みだ．この割り込みを受けると，しかたがないので今までやっていた仕事をそのままにして，その割り込み仕事を片づけ，またもとの仕事に戻っていく．場合によっては，割り込み仕事をしている最中に，また別の割り込み仕事が飛び込んでくることもあるかもしれない．そのうちには，最初になにをやっていたのかわすれてしまったりもする……かな？

逆に，この割り込みを積極的に使って効率的に仕事を進めている人もいる．たとえば，報告書をまとめていてコピーの必要が生じたとする．こういう場合，近くで暇そうにしている人間を見つけて，「ねぇ，これを10部コピーとってくれない？　終わったら教えてね」とやっておく．彼がコピーをしている間，自分は本来の仕事に専念していられるという寸法だ．「できました」という割り込みがかかると，それを受けとって資料の一部として綴じ込めばいい．

コンピュータにも，こういうしくみが用意されている．なにかのプログラムを実行している最中に「おい，ちょっと」と声をかけてそのプログラムを中断させ，あらかじめ決められている別のプログラムを実行させるというしくみだ．その直前まで実行していたプ

ログラムと，割り込みによって動作するプログラムとはなんの関係もないのが普通だ．したがって，割り込み処理に入る前に，割り込み処理が終わったときに確実に前のプログラムに復帰できるような手だてを講じておく必要がある．また，割り込み処理プログラムのほうは，もとのプログラムの実行に影響を与えるようなことをしてはならない．

人間の場合もそうだが，コンピュータはこの割り込みを拒否することやあと回しにすることができる．決められた時間のうちに一定の処理を実行しなければならないような場合は，その途中での割り込みを許しておくとその処理の分だけ実行終了が遅れてしまう．そういう場合は，一時的に割り込みがかからないようにしておいて，その代わりに割り込みの要求があったことをなんらかの方法で記録しておく．割り込みを起こした要因に待っていてもらうと考えてもいいだろう．こうしておけば，忙しい仕事が終わって割り込みを許可したときに待たされていた割り込みが入る．「ねぇあなた，隣の奥さんがね……」「ん？　あ，うん，ちょっと，え～と，よしと．ん，なんだっけ」「もういいわよっ」人間の世界では待ってくれない割り込みもある．

図1　割り込みの概念

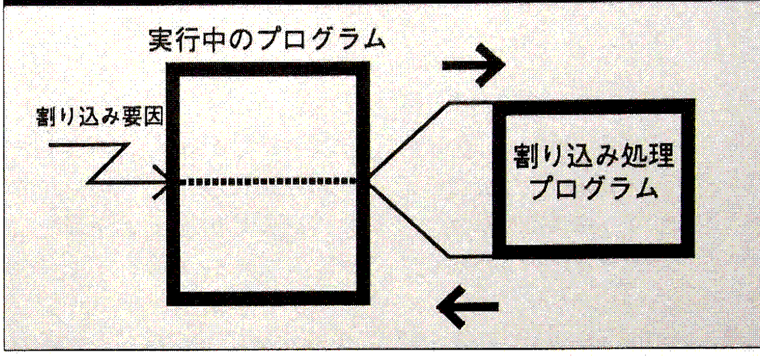
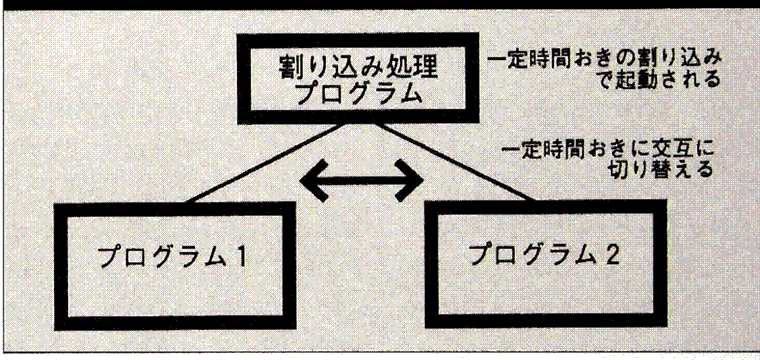


図2　マルチタスク



割り込み処理の効用

コンピュータにこういうしくみがあるとどういうことができるか，ないとどういう不都合があるか，それを考えてみることにしよう．

マルチタスクという言葉をご存じだろう．1台のコンピュータに同時にいろいろな仕事をさせる技術だ．同時にといってもプログラムはCPUによって実行されるので，CPUが1つしかなければ，ほんとうに同時に別々の仕事をさせるというわけにはいかない．しかし，たとえばいま，2つのプログラムがあったとして，この2つを非常に短い間隔で交互に少しずつ実行していくと見かけ上2つのプログラムが同時に動いているように見える．

この場合，2つのプログラムはそれぞれ別個のもので，おたがいのことは知らないのが普通だから，おたがいに譲りあって……などということは期待できない．なんらかの方法で外部から強制的に実行を切り替えてやる必要がある．このために割り込みを使う．ごく短い間隔で一定時間おきに割り込みが入るようにしておき，割り込み処理プログラムが，現在実行中のプログラムを交互に切り替えてやる．もちろん，切り替えるたびに最初から実行していたのではいつまでたっても先に進まないから，割り込み処理プログラムは，現在実行中のプログラムのレジスタなど実行環境をそっくりセーブしておき，別のところにセーブしてある別のプログラムの実行環境を戻してやる……という操作を行っている．

切り替えるプログラムは2つ以上でもいいが，同時に動くプログラムが増えるほど1つのプログラムの実行時間は長くかかることになる．それでも，たとえばキー入力待ちなどのようにあまりコンピュータがフルに動いていない状態のときに，別のプログラム，たとえば計算とか並べ替えのようなものを同時に動かせば，コンピュータを効率的に使うことができる．

こんなおおげさなものではなく，もっと身近なところでも割り込みが応用されている．急いでいるときなど，なにかのプログラムの実行中に次のコマンドのためのキー入力をすることがあるが，「先行入力」と呼ばれるこの機能も割り込みで処理されている．

たとえば，アセンブルを実行してい

るコンピュータはソースの解析にかかりきりで，キー入力などにかまっている暇はない．したがって，この間のキー入力などは無視されて当然なのだが，キー入力処理が割り込みで動作しているマシン（現在のパソコンはほとんどがそうだ）であればアセンブル中でもキー入力があるたびに割り込み処理プログラムが起動されて，入力されたキーデータを専用のバッファに格納しておいてくれる．アセンブルが終わってからMS-DOSがキーバッファを見にいくと，アセンブル中に入力された“LINKなんとかかんとか”というような文字列が書いてあるというわけだ．

次に，ないと困る場合を考えてみよう．通信ソフトがこの好例になる．かんたんな通信ソフトのロジックは図3のようになっている．

たとえば「受信データを処理する」という処理に，画面表示だけでなくデータをログファイルにも記録するという処理が入っていたとする．ここで，ディスクにこりこりと書いている間に次のデータがどんどん送られてきたらどうなるだろう．なにもしなければ，そのデータはまちがいなく捨てられてしまう．これを取りこぼさないためには，処理の最中に間断なく受信データが来ているかどうかを見ていなくてはならない．しかし，かんたんなものならそれも可能だろうが，少し複雑な通信ソフトではとてもじゃないがそんなことはしていられない．

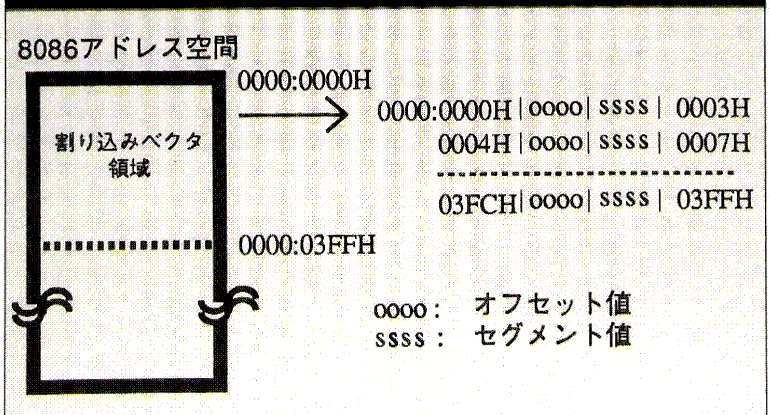
そこで，割り込み処理の登場だ．データが受信されるとCPUに割り込みが入るようにし，割り込み処理プログラムでそのデータをどこかのバッファに保存して，もとの処理に戻ればいいのである．受信データが存在した場合に行うべき処理は，このバッファにデータがあるかどうかを見て実行すればよいことになる．

このようにコンピュータにとって割り込みは，実に重要な役割を果たしているのだ．

図3　通信ソフトのロジック

```
begin
        if      受信データ有り？
                        then    受信データを処理する
        endif
        if      キー入力有り？
                        then    入力データを送信する
        endif
loop
```

図4　割り込みベクタ領域



8086の割り込み機構

さて，ここから8086の割り込みの機構を詳しく見ていくことにする．

前述のように，割り込みがかかると割り込み処理プログラムが起動されるが，そのためには割り込み処理プログラムのアドレスを，なんらかの方法でCPUが知る必要がある．また，割り込みが1つならば割り込み処理プログラムも1つでいいが，たくさんの割り込みを処理できるようにするためには，その数だけの割り込み処理プログラムが必要になり，それぞれの割り込みに対応した割り込み処理プログラムのアドレスを知らなくてはいけないことになる．

さらに，割り込み処理プログラムもプログラムの1つなので，8086CPUからアクセスできるメモリ空間に置かれている必要がある．8086ではメモリ上のアドレスをセグメント1ワード，オフセット1ワードの2ワード（4バイト）で管理しているので，割り込み処理プログラムもこういう形式のアドレスで管理される．

8086は全部で256種類の割り込みに対応できるように作られている．この256種類の割り込み処理プログラムのアドレスを記憶するためには，256×4＝1024バイトのデータが必要になる．8086では，このデータをCPU内部ではなく，割り込みベクタと呼ばれるメモリ上に置かれたテーブルに記憶している．このテーブルの置かれるアドレスは固定で，セグメント0，オフセット0からの1KBがそれにあてられている．

たとえば，割り込み番号3番の割り込みが入ると，CPUは割り込みベクタ

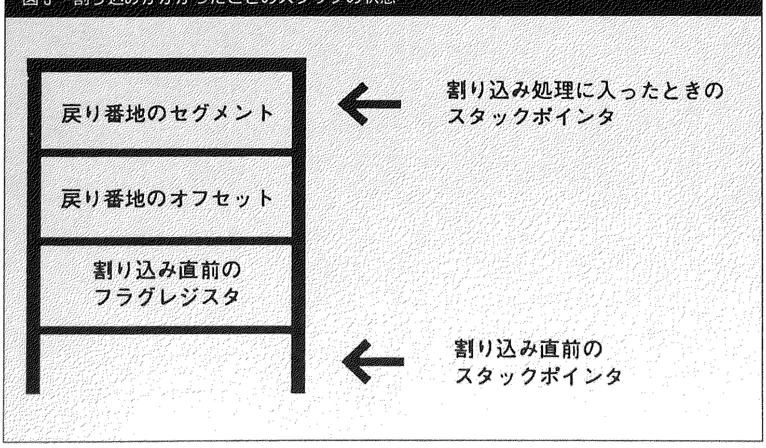
ワンラインNEWS
アスキーの西社長はソ連共産党機関紙プラウダで報じられたソ連製ソフトの詐取を否定した．同社は90年末にソ連でプログラムコンテストを行った．

の3番目，つまり3×4＝12（0CH）からの4バイトのデータをセグメントとオフセットの対として読み込み，そのアドレスに処理を渡すという動作をする．この領域がRAMになっている場合，その値はプログラムで自由に変更することができる．割り込みが入ったときにどこにある割り込み処理プログラムを起動するかは，プログラムの実行中にも変更が可能ということだ．TSR（メモリ常駐）プログラムなどは，このベクタ領域を書き換えて割り込み処理を自分自身で行うようにしているものが多い．ためしにあとで紹介するベクタ領域表示のプログラムを使って，PRINT.COM（MS-DOS標準のプリンタスプーラ）を起動する前とあとのベクタ領域を見て，このソフトがどのベクタを書き換えているのかを確認してみるといい（ツールの出力をそれぞれファイルにリダイレクトしておいて，あとでFCにかけるとよくわかる）．

さて，割り込み処理が終わったら，もとのプログラムに戻らなくてはならないから，現在実行中のプログラムのアドレスは保存しておく必要がある．また，割り込みが入るとCPUによって自動的に割り込み許可フラグがリセット（割り込みを受けつけない状態）されるので，直前の割り込み許可フラグの状態も保存しておく必要がある．

フラグレジスタ以外のデータを見ると，これはFar-Call時にスタックに積まれるデータと同じものである．これだけなら戻るときにはRETFを実行すればいいのだが，フラグレジスタも復帰しなければならないので，IRETという特別な命令を使ってもとのプログラムに復帰する．なお，ハードウェア割り込み処理は実際には，個々のハードウェアに依存する部分が多いので，プログラミングにあたってはシステム全体に関する知識が必要となる．

図5　割り込みがかかったときのスタックの状態



ソフトウェア割り込み

ところで，8086にはいままで見てきた割り込みのほかにもうひとつ，「ソフトウェア割り込み」というものがある．本来ハードウェアが主体になって動いているはずの割り込みをソフトウェアで行う……ちょっと変な感じがするが，実はこのソフトウェア割り込みというのは，割り込みがかかってあらかじめ決められたベクタを参照して，そのアドレスに飛んでいくという一連の動きをソフトウェアでシミュレートするもので，かんたんにいうとサブルーチンコールの一種なのだ．割り込みベクタもハードウェア割り込みと同じ領域を使用する．

ソフトウェア割り込みの例としては，おなじみのMS-DOSシステムコール，INT21Hがこれに該当する．ソフトウェア割り込みはハードウェアによって引き起こされるものと違って，プログラマが明示的にINTという命令を記述して使用する．

この方法でサブルーチンコールのインターフェースを作ると，そのアドレスをベクタテーブルにセットさえしてやれば，処理プログラムの本体はどこにあってもいいことになる．呼び出すほうも，実際の処理プログラムがどこにあるかを知る必要はなく，ただ「この処理をする場合は21H番の割り込みを使えばいい」ということを覚えているだけでいい．システムの都合で，実際の処理プログラムの置かれているアドレスが変更になってもいっこうにかまわないわけである．

以上，ハードウェア割り込みとソフトウェア割り込みについて調べてきたが，どちらもシステムの動作にとってたいへん重要なものであることがおわかりいただけるだろう．とくにハードウェア割り込みは，私たちがまったく気がつかないところでシステムをしっかりと支えてくれている．そして，この割り込み動作のすべてが「割り込みベクタ」というたった1KBほどのテーブルに握られているのである．

今月のプログラム

この重要な割り込みベクタを表示・設定するプログラムを作ってみる．割り込み処理プログラムがどこに置かれているかを調べるには，デバッガを起動してベクタをダンプしてみるのが普通だが，このプログラムを使えばコマ

ンドラインオプションで割り込みベクタのオフセットを指定するだけでその内容を表示することができる．オフセットではなく，割り込み番号で指定することもでき，「ここからここまで」という範囲指定も可能である．

割り込みベクタを設定する機能はあまり使わないかもしれないが，自作のTSRプログラムの試験などで，一時的に動作を止めるような場合に利用できるだろう．こちらもオフセットだけでなくベクタ番号や範囲指定も可能だが，まちがってシステムが使っているベクタを書き換えることのないように十分注意してもらいたい．

使い方

■表示

A> VECT

ベクタ領域をすべて表示する．表示形式は図6の通り．

先頭の4桁の数字は割り込みベクタのオフセット（セグメントは0に固定されているので表示しない），次のカッコつきの4桁の数字は割り込み番号で，単にオフセットを4で割ったものだ．そのあと一行にssss:ooooの組み合わせが4個ずつ表示される．最初がセグメントでコロンのあとがオフセットなのでまちがえないように．

A> VECT 84

INT21Hのベクタを表示する（数値は十六進として評価される）．

A> VECT #21

INT21Hのベクタを表示する（ベクタ番号指定）．

A> VECT #3-#7

INT3からINT7の間のベクタを表示する．

図6　ベクタ表示プログラムの表示形式

```
0000(0000)  F900:02FD FD80:0936 FD80:078C 26B1:01FB
```

COLUMN　REPNE＋SCASBの誤解

8086には，文字列を扱うのに非常に便利な命令が用意されている．SCAS, MOVS, STOSなどの'S'つきの命令群だ．これに繰り返しのためのREP族のプリフィクスを組み合わせて使うと，文字列の移動や文字の検索が，少ないコード量で簡潔に記述できる．

たとえば，プログラム1のような一連の命令群が，プログラム2のように置き換えられる（ただしプログラム1は厳密な意味ではプログラム2と等価ではない）．

たいへん便利な命令なのだが，アセンブラを勉強しはじめたころどうしてもわからないことがあった．いろいろなプログラムを参考にしたのだが，SCASBに入る前にZフラグをリセット（NZに）しているプログラムと，していないプログラムがあるのだ．NZにするのがほんとうで，理由はREPNEが最初にZフラグを検査するからだと考えてみたのだが，比較命令より先にZフラグを検査するというのはおかしな話だ．

どうにも気持ちが悪いのでいろいろと資料をあさっていたら，ちょっと古い本にREPプリフィクスの詳しい説明が載っているのを見つけた．それはこういうものだった（REPNE　SCASBを例にとる）．

①まずCXの内容を検査する．CXが0ならばSCASB命令をスキップする（つまりこの一連の処理を終了する）．
②SCASB命令を1回実行する．
③CXをデクリメントする．
④SCASBの結果がZならばこの一連の処理を終了し，NZならば①へ．

これでようやくわかった．①の処理ではフラグに影響が出ないので，最初からCXが0で入ってきた場合にはZフラグの値がそのままになっているのだ．だから最初にNZ，つまり「見つからなかった」ということにしておくのだ．CXが必ず0以外であるという保証があるのならば，SCASBは必ず1回は実行されるからこういう処置（ZフラグをNZの状態にする）はなくてもいい．

ということで，CLDの状態でのREPNE　SCASBを順序はともかく，外部仕様をあわせて基本的な命令群に置き換えてみると，プログラム3のようになりそうだ（割り込みに対する応答は考慮していない）．

……と書いて得々としていたら，後ろから「このほうが短い」とアセンブラ仙人の声（プログラム4）．

まいりました（とほほ）．

プログラム1

```
LP:
        CMP     ES:[DI],AL
        JZ      DONE
        INC     DI
        DEC     CX
        JNZ     LP
        CMP     CX,1      ; NZ
DONE:
```

プログラム2

```
        REPNE   SCASB
```

プログラム3

```
LP:     JCXZ    DONE      ; フラグは不変
        INC     DI
        DEC     CX
        CMP     AL,ES:[DI - 1]
        JNZ     LP
DONE:
```

プログラム4

```
        JCXZ    DONE      ; フラグは不変
LP:     INC     DI
        CMP     AL,ES:[DI - 1]
        LOOPNZ  LP        ; Thanks Charon
DONE:
```


ワンラインNEWS
1MビットDRAMの価格下落に歯止めがかかってきた．しかしこれは減産による一時的な市況安定といわれている．各社の製造体制は4Mビットに移行している．

■設定

A> VECT #80=1234:5678

INT80Hのベクタに1234:5678Hをセットする.

※セットするデータは必ずセグメント:オフセットの形であること.

A> VECT #80-#83=0:0

INT80HからINT83Hの間のベクタをすべて0:0にセットする.

■そのほか

いくつかの指定を空白を挟んで続けることもできる.

A> VECT #80-#87 #81-86=0:0 #80-87

この機能があるため, 1つのコマンドの中には空白を挟んではいけない.

A> VECT #80 - #87

このように, ベクタ番号を指定する文字列に空白を入れたりするのはまちがいだ. なお, ベクタ番号の指定がまちがっていたりデータが決められたフォーマットを守っていない場合はなにもせずに終了する.

プログラムの解説

ではプログラム (リスト1) の細かな解説をしよう.

■main

352行—356行

まずコマンドラインオプションを解析する.

360行—365行

オプションが1つもなければベクタ領域を全部ダンプする処理に入る.

368行—372行

オプションがあれば, その個数だけ処理する. 途中でエラーが発生したらそれ以降の処理を打ち切って終了する.

■vect_sub, 処理本体

307行—310行

ベクタオフセットあるいは番号を取得し, それを開始ベクタと終了ベクタに入れる.

312行—326行

数値の次が'-'ならば範囲指定なので, 次のベクタオフセットまたは番号を取得して終了ベクタに入れる.

331行—334行

次が'='ならばベクタの設定, コマンドの終りならば表示処理へ.

■vct_set, ベクタセット処理

274行

ベクタに設定する値を取得する.

282行—285行

開始ベクタから終了ベクタまでをその値にセットする. ベクタのセットはDOSのファンクションコールを使用する.

■vct_dump, ベクタダンプ処理

200行—212行

ヘッダ部分の文字列を作成する. フォーマットは "oooo (nnnn)".

219行—221行

システムコールでベクタ値を取得する.

228行—234行

ベクタ値の文字列を作成する. フォーマットは "ssss:oooo ".

■そのほか

これ以外の小さな処理は, すべて上記処理のためのサブルーチンである. 詳細はソースリストのコメントを参照してもらいたい.

ベクタ値の設定や取得は別にDOSのファンクションコールを使わなくてもできるのだが, あえてこのようにした. たぶんこういうファンクションコールが用意されているのは, マイクロソフトの将来への布石だろうと思われるので, そういうものには逆らわないほうがよかろうと判断したのである. 多少冗長でもツールは長生きしたほうがいい.

リスト1

```
 1: TITLE   Command VECT.EXE
 2:
 3: PAGE    60,128
 4:
 5: .xlist
 6: include         console.h
 7: include         dosfunc.h
 8: .list
 9:
10: _TEXT   SEGMENT  WORD PUBLIC 'CODE'
11: _TEXT   ENDS
12:
13: _DATA   SEGMENT  WORD PUBLIC 'DATA'
14: _DATA   ENDS
15:
16: _STACK  SEGMENT STACK
17: _STACK  ENDS
18:
19: DGROUP  GROUP   _DATA,_STACK
20:         ASSUME DS:DGROUP, SS:DGROUP
21: ;
22: ;
23: ; ======================== 定数宣言 ===========
24: ;
25: PSP_PARAM       equ     80h             ; PSPのコマンドラインオプション
26:                                         ; のオフセット。
27: ARGMAX          equ     16              ; コマンドラインオプションの最大数。
28: ARGBUFSIZE      equ     256             ; コマンドラインオプション処理のための
29:                                         ; バッファのサイズ。(本文参照)
30: ;
31: VSTART          equ     00H             ; ベクタ番号の最小値と
32: VEND            equ     0ffh            ; 最大値。
```

```
;
; ================================================
;
_TEXT       SEGMENT
        ASSUME  CS: _TEXT
;
extrn           printmes:near,putchar:near,print_asciz:near
extrn           set_param:near,skip_space:near
;
; -------------------- 十六進文字列をバイナリ値に変換
;
; 入力レジスタ  DS:SI=処理する文字列の先頭アドレス
; 出力レジスタ  DS:SI=最後に変換したデータのつぎのアドレス
;               BX=変換した値
;               CY:0=変換成功,CY:1=変換失敗
;
get_hvalue      proc
        push    ax
        push    cx
        mov     bx,0            ; まず結果を０にし
        mov     cx,bx           ; 処理個数をリセットする。
gth_lp:
        mov     al,[si]
        cmp     al,'a'          ; 小文字－＞大文字変換。
        jb      gth0
        cmp     al,'f'
        ja      gth0
        and     al,11011111b    ; 'a'-'f' --> 'A'-'F'
gth0:
        sub     al,'0'
        jc      gth_end         ; '0'より小さいのは数字ではない。
        cmp     al,9            ; '0'を引いて9までなら数字。
        jbe     gth_next
        sub     al,7            ; '9' ～ 'A'
        cmp     al,0ah          ; 10より小さければ数字ではない。
        jb      gth_end
        cmp     al,0fh          ; 15以下ならば数字である。
        jbe     gth_next
gth_end:
        cmp     cx,1            ; 数字がひとつもなければエラー。
gth_end0:
        pop     cx
        pop     ax
        ret
gth_next:
        cmp     bh,10h
        cmc
        jc      gth_end0
        shl     bx,1            ; 4ビット左シフト（１６倍）
        shl     bx,1
        shl     bx,1
        shl     bx,1
        or      bl,al           ; 結果に加えて
        inc     si              ; ポインタと
        inc     cx              ; カウンタを更新して
        jmp     gth_lp          ; もう一度。
get_hvalue      endp
;
; -------------------- ssss:oooo形式の十六進文字列をバイナリ値に変換
;
; 入力レジスタ  DS:SI=処理する文字列の先頭アドレス
; 出力レジスタ  DS:SI=最後に変換したデータのつぎのアドレス
;               AX:BX=変換した値
;               CY:0=変換成功,CY:1=変換失敗
;
get_vvalue      proc
        call    get_hvalue
        jc      gtvv_end
        cmp     byte ptr [si],':'
        stc
        jnz     gtvv_end        ; ssss:ooooの形式でなければエラー。
        inc     si
        mov     ax,bx
        call    get_hvalue
gtvv_end:
        ret
get_vvalue      endp
;
;
; -------------------- ベクタオフセット／番号をバイナリ値に変換
;
; 入力レジスタ  DS:SI=処理する文字列の先頭アドレス
; 出力レジスタ  DS:SI=最後に変換したデータのつぎのアドレス
;               BX=変換した値
;               CY:0=変換成功,CY:1=変換失敗
;
get_vnumber     proc
        cmp     byte ptr[si],'#'
        jnz     gtvn0
        inc     si              ; 先頭が'#'ならばベクタ番号指定。
        call    get_hvalue
        jc      getvn_end
        jmp     gtvn1
gtvn0:
        call    get_hvalue
        jc      getvn_end
        shr     bx,1            ; オフセット指定は４で割って番号に変換。
        shr     bx,1
gtvn1:
        cmp     bx,VEND + 1
        cmc
getvn_end:
        ret
get_vnumber     endp
;
; -------------------- 数値を４桁の十六進文字列に変換
;
; 入力レジスタ  ES:DI=文字列を格納する先頭アドレス
;               AX=変換する値
; 出力レジスタ  ES:DI=最後に格納した文字列のつぎのアドレス
; 破壊レジスタ  AX
;
conv_hword      proc
        xchg    al,ah
        call    conv_hbyte      ; ＣＡＬＬが２回必要なところを
        xchg    al,ah           ; １回ですませてしまう。
conv_hbyte:
        push    ax
        shr     al,1
        shr     al,1
        shr     al,1
        shr     al,1
        call    conv_hnibble    ; ここもそう。
        pop     ax
        and     al,0fh
conv_hnibble:
        add     al,'0'
        cmp     al,'9'
        jbe     cvh0
        add     al,7
cvh0:
        stosb
        ret                             ; RETはここ一ヶ所だけ!!。
conv_hword      endp
;
; -------------------- 表示バッファの出力処理
;
; 入力レジスタ  なし
; 出力レジスタ  なし
;
flush_buffer    proc
        push    si
        mov     si,offset dump_buffer
        call    print_asciz
        PRINTM  H_STDOUT,<0dh,0ah>
        pop     di
        ret
flush_buffer    endp
;
; -------------------- ベクタ内容の表示
;
; 入力レジスタ  なし
; 出力レジスタ  なし
;
vct_dump        proc
        push    ax
        push    bx
        push    cx
        push    dx
        push    di
        mov     bx,[vect_start]
vctd_lp:
        mov     di,offset dump_buffer
        cmp     bx,[vect_end]
        ja      vctd_end
        mov     cx,4
        mov     ax,bx
        push    ax
        shl     ax,1
        shl     ax,1
        call    conv_hword
        mov     al,'('
        stosb
        pop     ax
        call    conv_hword
        mov     al,')'
        stosb
        mov     al,' '
        stosb
        stosb
vctd_lp0:
        cmp     bx,[vect_end]
        ja      vctd_end
;
        push    es
        push    bx
        mov     ah,DOS_GETVECT
        mov     al,bl
        int     21h
        mov     dx,es
        mov     ax,bx
        pop     bx
        pop     es
;
        xchg    ax,dx
        call    conv_hword
        mov     al,':'
        stosb
        xchg    ax,dx
        call    conv_hword
        mov     al,' '
        stosb
        inc     bx
;
        loop    vctd_lp0
        mov     al,STERM
        stosb
        call    flush_buffer
        jmp     vctd_lp
vctd_end:
        cmp     di,offset dump_buffer
        jz      vctd0
        mov     al,STERM
        stosb
        call    flush_buffer
vctd0:
        pop     di
        pop     dx
        pop     cx
        pop     bx
```

```
         pop     ax
         ret
vct_dump         endp
;
; -------------------- ベクタ内容の設定
;
; 入力レジスタ  なし
; 出力レジスタ  なし
;
vct_set          proc
         push    ax
         push    bx
         push    cx
         push    dx
         push    ds
         mov     cx,[vect_start]
         mov     ax,cx
         mov     cx,[vect_end]
         cmp     cx,ax
         jc      vcts_end
         push    ax
         call    get_vvalue
         jc      vcts_end
         mov     ds,ax
         mov     dx,bx
         pop     ax
         jc      vcts_end
         mov     ah,DOS_SETVECT
vcts_lp:
         int     21h
         inc     al
         cmp     al,cl
         jbe     vcts_lp
vcts_end:
         pop     ds
         pop     dx
         pop     cx
         pop     bx
         pop     ax
         ret
vct_set          endp
;
; ------------------- 実質的なメイン
;
; 入力レジスタ  DS:BX=処理するコマンドの先頭アドレス。
; 出力レジスタ  CY:0=処理成功,CY:1=処理失敗
;
vect_sub         proc
         push    ax
         push    bx
         push    cx
         push    si
;
         mov     si,bx
         call    get_vnumber
         jc      vct_end0
         mov     [vect_start],bx
         mov     [vect_end],bx
         lodsb
         cmp     al,'-'
         jnz     vct1
;
if       0                        ; テスト用。普段は抑止しておく。
         mov     al,[si]
         cmp     al,STERM
         jnz     vct0
         mov     word ptr[vect_end],VEND
         jmp     vct4
endif
;
vct0:
         call    get_vnumber
         jc      vct_end0
         mov     [vect_end],bx
         lodsb
         cmp     al,STERM
         jz      vct4
vct1:
         cmp     al,'='
         jnz     vct3
vct2:
         call    vct_set
         jmp     vct_end0
vct3:
         cmp     al,STERM
         stc
         jnz     vct_end0
vct4:
         call    vct_dump
vct_end0:
         pop     si
         pop     cx
         pop     bx
         pop     ax
         ret
vect_sub         endp
;
main             proc
         mov     si,PSP_PARAM + 1         ; DSはPSPを指している。
         mov     ax,DGROUP
         mov     es,ax
         mov     di,offset param
         mov     cx,ARGMAX
         call    set_param
         mov     ax,es
         mov     ds,ax
         mov     si,offset param
         cmp     cx,0
         jnz     main_loop
;
         mov     word ptr[vect_start],VSTART
         mov     word ptr[vect_end],VEND
         call    vct_dump
         jmp     main_ende
main_loop:
         lodsw
         mov     bx,ax
         call    vect_sub
         jc      main_ende
         loop    main_loop
main_end:
         mov     al,NORMAL_END    ; 終了コードを返して終わり。
main_ende:
         mov     ah,DOS_TERMINATE
         int     21h
main     endp

_TEXT    ENDS

_STACK   SEGMENT STACK
;
                 dw      1024          dup (?)
;
_STACK   ENDS

_DATA    SEGMENT  WORD PUBLIC 'DATA'
;
param            dw      ARGMAX        dup (?)
                 db      ARGBUFSIZE    dup (?)
;
vect_start       dw      (?)
vect_end         dw      (?)
;
dump_buffer      db      80            dup (?)
;
_DATA    ENDS

         END     main
```

リスト2

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        console.h        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; ======================== 定数宣言 ================
;
STERM            equ     00H          ; ASCIZ文字列の終端子。
CR               equ     0dh
;
; ==================================================
;
;
; ======================== マクロ定義 =============
;
PRINTM           macro   handle,arg
                 local   prtdt
         push    bx
         mov     bx,handle
         call    printmes
         dw      prtdt - $ - 2
         db      arg
prtdt:
         pop     bx
ENDM

;
; ==================================================
```

リスト3

```
;;;;;;;;;;;;;;;;;;;;;;
;     dosfunc.h      ;
;;;;;;;;;;;;;;;;;;;;;;
;
H_STDOUT         equ     1            ; 標準出力ハンドル
H_ERROR          equ     2            ; 標準エラー出力ハンドル
;
NORMAL_END       equ     0            ; プロセス正常終了コード
;
AT_A             equ     00100000b    ; アーカイブ属性
AT_D             equ     00010000b    ; サブディレクトリ
AT_V             equ     00001000b    ; ボリュームラベル
AT_S             equ     00000100b    ; システムファイル
AT_H             equ     00000010b    ; 隠しファイル
AT_R             equ     00000001b    ; リードオンリーファイル
;
ATTR_ALL         equ     AT_A + AT_D + AT_V + AT_S + AT_H + AT_R
;
; === INT-21H機能番号 ===
;
DOS_SETDTA       equ     1ah          ; DTAアドレス設定
DOS_SETVECT      equ     25h          ; 割り込みベクタ値の設定
DOS_GETVECT      equ     35h          ; 割り込みベクタ値の取得
DOS_HWRITE       equ     40h          ; ファイル/デバイスへの出力
DOS_TERMINATE    equ     4ch          ; プロセス終了
DOS_FINDFIRST    equ     4eh          ; 最初のファイルの検索
DOS_FINDNEXT     equ     4fh          ; つぎのファイルの検索
```

ワンラインNEWS
富士通はパソコンLAN用OS「NetWare386 Ver.3.1J」を発売した。この製品はノベル社からOEM調達したもの。

アセンブルとリンク

表示ルーチンは先月号の“CONSOLE. ASM”に1つ処理を加えたものを使う．また，“SETPARAM. ASM”にコマンドラインオプションの処理を独立させたので，これもアセンブルしてリンクしなければならない．また，ヘッダファイルにも新しい情報が増えているので注意してほしい(リスト2，3，4参照)．

A> MASM CONSOLE;

A> MASM SETPARAM;

A> MASM VECT;

A> LINK VECT+SETPARAM+CONSOLE;

※自作のプログラムから“set_param”を使用する場合も，おなじように自分のプログラムのオブジェクトファイルといっしょに“SETPARAM”のオブジェクトを“+”でつないで指定すればいい．なお呼び出すほうのプログラムに“extrn”宣言をわすれないこと．

リスト4

```
TITLE   console.asm コンソール入出力処理
PAGE    60,128
include         console.h
include         dosfunc.h
_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
_STACK  SEGMENT STACK
_STACK  ENDS
DGROUP  GROUP   _DATA,_STACK
        ASSUME DS: DGROUP, SS: DGROUP
_TEXT      SEGMENT
        ASSUME  CS: _TEXT
;
public          printmes,putchar,print_asciz
public          color_normal,color_red,color_green,color_yellow
public          color_blue,color_magenta,color_cyan,color_white
;
; -------------------- 標準出力への出力機能
;
; === ＡＳＣＩＺ文字列を出力 ===
;
print_asciz     proc
        push    ax
        push    bx
        push    cx
        push    dx
        push    di
        push    es
;
        mov     di,si           ; 文字列のアドレスは ds:si で渡される。
        mov     ax,ds           ; それを es:di にセットする。
        mov     es,ax
        mov     al,0
        mov     cx,65535
        push    di
        repne scasb             ; STERMを探して表示するサイズを計算する。
        pop     di
        not     cx
        dec     cx
        mov     dx,di
        mov     bx,H_STDOUT
        mov     ah,DOS_HWRITE
        int     21h
;
        pop     es
        pop     di
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
print_asciz     endp
;
; === 戻り番地のつぎのアドレスからのデータを出力 ===
;
printmes        proc
        push    ax
        push    cx
        push    dx
        push    ds
        push    si
        push    bp
        mov     bp,sp
;
        mov     si,[bp + 12]    ; 戻りアドレスを取得して
        mov     cx,cs:[si]      ; そこから表示データのバイト数を得る。
        add     si,2            ; 表示データアドレスを指して
;
        mov     dx,si           ; それをＤＸにも設定し
        add     si,cx           ; 最後のデータのつぎを指して
        mov     [bp + 12],si    ; それを戻り番地とする。
;
        mov     ax,cs           ; コードセグメント上のデータなので、
        mov     ds,ax           ; それをＤＳにセットし（ＤＸはセットずみ）
        mov     ah,DOS_HWRITE   ; 標準出力に
        int     21h             ; 出力する。
;
        pop     bp
        pop     si
        pop     ds
        pop     dx
        pop     cx
        pop     ax
        ret                             ; このとき戻りアドレスは更新されている。
printmes        endp
;
; === 一文字出力 ===
;
putchar         proc
        push    bx
        push    cx
        push    dx
        push    ds
;
        push    ax              ; 表示データをＰＵＳＨする。
;
        mov     dx,sp           ; データはスタックの上にある。
        mov     ax,ss
        mov     ds,ax
        mov     cx,1
        mov     bx,H_STDOUT     ; 本当はもっと簡単なファンクションが
        mov     ah,DOS_HWRITE   ; あるのだが、出力を統一したかったので。
        int     21h
;
        pop     ax
;
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        ret
putchar         endp
;
; ------------------- エスケープシーケンスによる表示色変更処理。
;                     エラーハンドルに出力することで、リダイレ
;                     クト時にファイルにエスケープシーケンスが
;                     残ってしまうことを避けている。もし残した
;                     いならば、"H_ERROR"を"H_STDOUT"に変えること。
;
color_normal    proc
        PRINTM  H_ERROR,<1bh,'[0m'>
        ret
color_normal    endp
color_red       proc
        PRINTM  H_ERROR,<1bh,'[31m'>
        ret
color_red       endp
color_green     proc
        PRINTM  H_ERROR,<1bh,'[32m'>
        ret
color_green     endp
color_yellow    proc
        PRINTM  H_ERROR,<1bh,'[33m'>
        ret
color_yellow    endp
color_blue      proc
        PRINTM  H_ERROR,<1bh,'[34m'>
        ret
color_blue      endp
color_magenta   proc
        PRINTM  H_ERROR,<1bh,'[35m'>
        ret
color_magenta   endp
color_cyan      proc
        PRINTM  H_ERROR,<1bh,'[36m'>
        ret
color_cyan      endp
color_white     proc
        PRINTM  H_ERROR,<1bh,'[37m'>
        ret
color_white     endp
_TEXT   ENDS
        END
```


ワンラインNEWS
ワープロの出荷台数が伸び悩んでいる．高機能を搭載し，価格を15万～20万円に維持したため，ブックパソコンと価格がぶつかり需要が流れたらしい．

ヘッダファイルを除いてもファイル数が3個になった．そろそろ限界である．これ以上ファイルが増えると，MAKEでも使わないと体はともかく頭がもたない．

今月のお勧め

コマンドラインオプションの処理は，アセンブラでツールを書く場合にめんどうなもののひとつだ．Cだとライブラリのスタートアップ処理で解析をしてテーブルを作りポインタを渡してくれるのだが，アセンブラではそれも全部自前でやらなければならない．

しかたがないので作っておこう．この処理は汎用なので，一度作っておけば毎回それを使うことができる．Cのようにファイル名のワイルドカードの展開まではやっていないが，空白で区切られたコマンドラインオプションを1つ1つASCIZ文字列（最後に00Hが書かれた不定長の文字列）に変換してオフセットテーブルにセットする．空白の入ったデータはシングル／ダブルクォートで囲めばいいようにしてある．なお，DS:SIで渡されるコマンドラインオプションの最後には0DHが書かれていることを期待しているので，ASCIZ文字列を渡す場合は注意してほしい．

データ構造

図7のように，まずバッファ領域を確保し，DS:SI，ES:DIとCXにデータをセットして“set_param”を呼び出す．戻ってきたときにはCXに実際に処理したコマンドラインオプションの数が入っている．コマンドラインオプションの数がCXで指定した値以上であった場合には，多い分は捨てられる．ASCIZ文字列領域のサイズは検査していないので，多めに確保しておくこと．といってもコマンドラインオプションが最大127バイトだから，それにテーブルの要素数につき1個の終端子（00H）のぶんを加えただけでいい．

なお，DSとESはプログラムの起動時にはPSPセグメントを指しているので，DSはそのままにしてESが自分のデータセグメントを指すようにして使えばいい．

この部分は抜き出して別ファイルにしてあるので，自分で作ったプログラムとリンクして使える．リンクのしかたは「アセンブルとリンク」の項を参考にしてほしい．

使用例

使用例を図8に示す．

たとえばいま，コマンドラインオプ

図7　データ構造

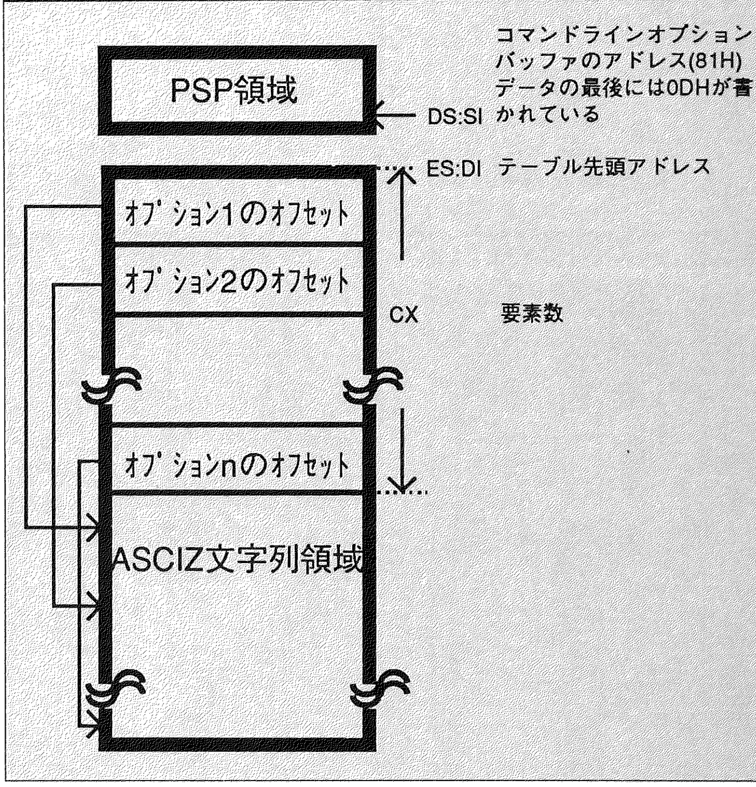


図8　set_paramの使用例

```
ARGMAX          equ     16              ; コマンドラインオプションの最大数．
ARGBUFSIZE      equ     256             ; コマンドラインオプション処理のための
                                        ; バッファのサイズ．
_CODE   SEGMENT
;
extrn           set_param:near          ; この宣言を忘れないこと．
;
        mov     si,PSP_PARAM + 1        ; 81H．DSはPSPを指している．
        mov     ax,DGROUP
        mov     es,ax                   ; ESは自分のデータセグメントを指し
        mov     di,offset param         ; DIはデータ領域のオフセットを指す．
        mov     cx,ARGMAX               ; 最大処理個数をセットして
;
        call    set_param               ; 呼び出す．
;
        ..........                      ; CXに処理した個数が入っている．
_CODE   ENDS

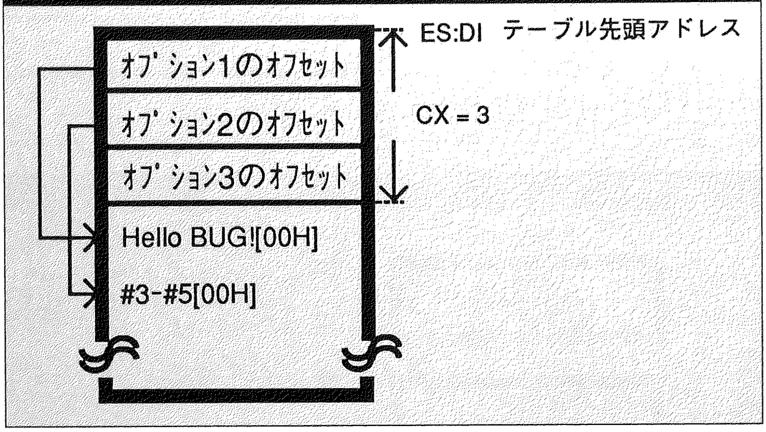
_DATA   SEGMENT WORD PUBLIC 'DATA'
param           dw      ARGMAX          dup (?)
                db      ARGBUFSIZE      dup (?)
_DATA   ENDS
```

ションに以下のような文字列が入っていたとしよう.

"Hello BUG!"　#3-#5［0DH］

この状態で各レジスタをセットして，要素数3で"set_param"を呼び出すと，戻ってきたときにはバッファには図9のようにデータがセットされている.

図9　バッファの状態



今月の反省

これからもたぶんそうなりそうだが，前月に掲載したソースに加除訂正したものを掲載することになってしまった. 前もってしっかりしたライブラリでも作っておいて話をはじめたほうがいいのだろうが，そうなると連載の最初はライブラリの説明だけになってしまい，書いているほうとしてはちっともおもしろくない. 書いているほうがそうなんだから,それを読まされる読者はもっとおもしろくなかろう.しばらくやっていくうちにだんだん整備されていくので,どうかひとつご勘弁願いたい.

リスト5

```
TITLE   set_param

PAGE    60,128

include         console.h

_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS

_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS

_STACK  SEGMENT STACK
_STACK  ENDS

DGROUP  GROUP   _DATA,_STACK
        ASSUME DS: DGROUP, SS: DGROUP

_TEXT      SEGMENT
        ASSUME  CS: _TEXT
;
public          skip_space,set_param
;
; ------------------- 空白のスキップ
;
; 入力レジスタ  DS:SI=文字列の先頭アドレス
; 出力レジスタ  AL=空白以外の文字,DS:SI=空白以外の文字のアドレス
;
skip_space      proc
        mov     ax,[si]
        cmp     al,' '
        jz      skps_nb
        cmp     ax,'　'
        jz      skps_nw
        ret
skps_nw:
        inc     si
skps_nb:
        inc     si
        jmp     skip_space
skip_space      endp
;
; ------------------- コマンドラインオプションの解析とセット（本文参照）
;
; 入力レジスタ  DS:SI=コマンドラインオプションの先頭アドレス
;               ES:DI=処理バッファの先頭アドレス
;               CX=オフセットテーブルの要素数（注意：バイト数ではない）
; 出力レジスタ  CX=処理したコマンドラインオプションの数
;
set_param       proc
        push    ax
        push    bx
        push    dx
        push    si
        push    di
        mov     dx,cx           ; 要素数をＤＸと
        mov     bx,cx           ; ＣＸにセットし
        inc     dx
        shl     bx,1            ; 要素数をサイズに変換。
setp_lp:
        dec     dx              ; 要素数を超えるなら
        jz      setp_end        ; 終了。
        call    skip_space
        cmp     al,CR           ; データ終わり
        jz      setp_end        ; ならば終了
        mov     ax,di           ; つぎのコマンドラインオプションを
        add     ax,bx           ; セットするアドレスを
        stosw                   ; テーブルにセットして
        sub     bx,2            ; オフセットを調整。
setc_lp:
        mov     ax,[si]
        cmp     al,CR           ; データ終わりか、
        jz      setp_end
        cmp     al,' '          ; 半角か
        jz      setc_end
        cmp     ax,'　'         ; 全角の空白を見つけたらつぎのオプションを処理
        jz      setc_end
        cmp     al,'"'          ; シングル／ダブルクォート文字なら
        jz      setp_q          ; その処理へ。
        cmp     al,"'"
        jz      setp_q
;
        mov     es:[di + bx],al ; それ以外の文字はコピーして
        inc     bx
        inc     si
        jmp     setc_lp         ; 繰り返す。
;
setc_end:
        mov     byte ptr es:[di + bx],STERM
        inc     bx              ; オプション文字列を終端して
        jmp     setp_lp         ; つぎのオプション文字列の処理へ。
;
setp_q:                         ; クォート文字列の処理。
        xchg    ah,al
        inc     si
setp_qlp:
        lodsb
        cmp     al,CR           ; クォートが終端されていなかったら
        jz      setp_end        ; そのまま終了する。
        cmp     al,ah
        jnz     setp_q0
        mov     byte ptr es:[di + bx],STERM
        inc     bx              ; クォートが終端されたら
        jmp     setp_lp         ; つぎのオプション文字列の処理へ。
setp_q0:
        mov     es:[di + bx],al ; クォート内の文字を一つコピーして
        inc     bx
        jmp     setp_qlp        ; 繰り返す。
;
setp_end:
        mov     byte ptr es:[di + bx],STERM
        mov     cx,di
        pop     di
        pop     si
        pop     dx
        pop     bx
        pop     ax
        sub     cx,di           ; 処理したオプションの数を
        shr     cx,1            ; ＣＸにセットする。
        ret
set_param       endp

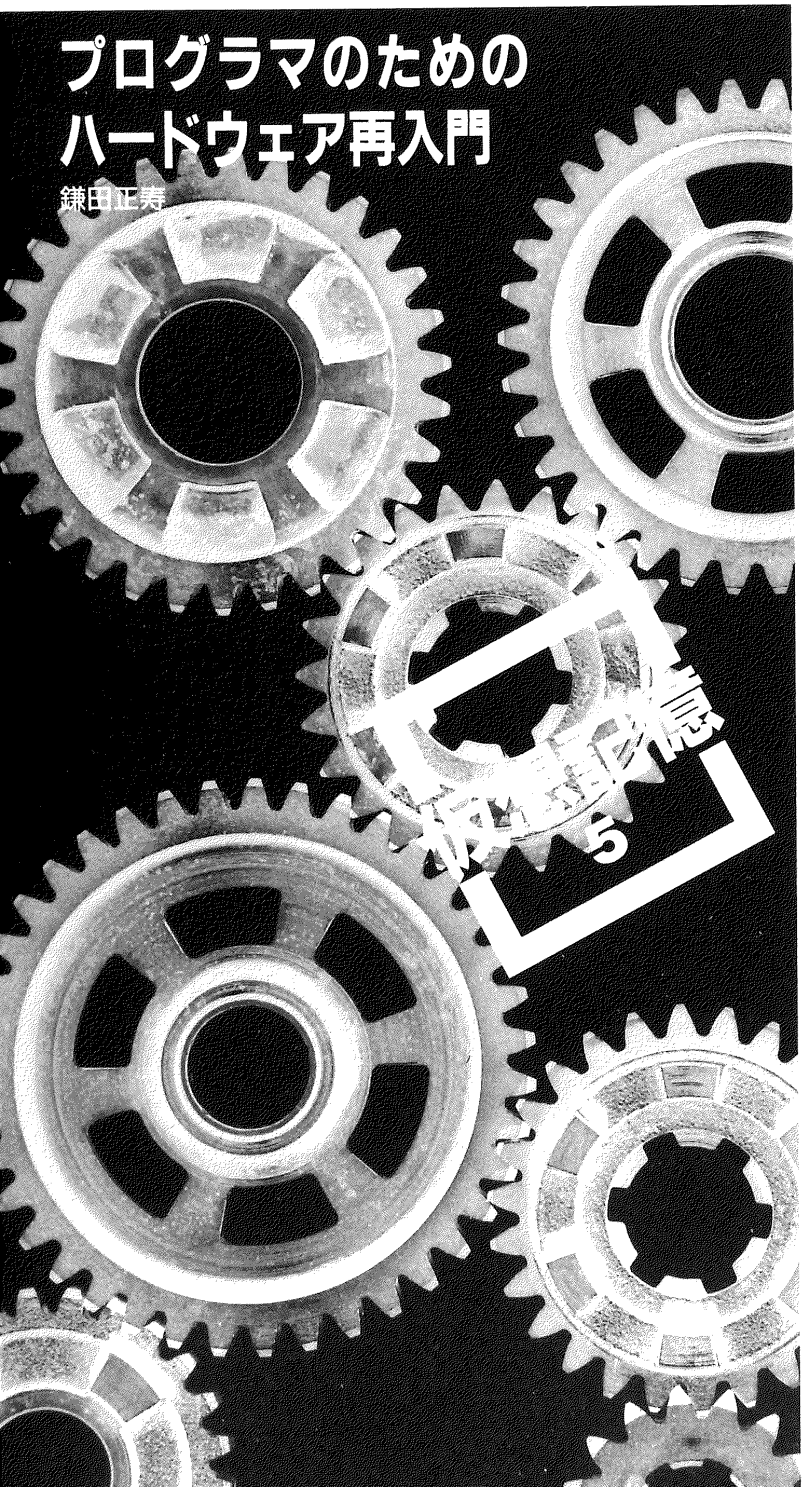
_TEXT   ENDS

        END
```

Reintroduction to HARDWARES

プログラマのための ハードウェア再入門

鎌田正寿



仮想記憶は，本来は少ない実装メモリよりも大きなメモリ空間を提供する技術として生まれたものです．過去においてメモリは高価で，1MB程度でも多くのメモリチップを必要としました．しかし，現在では1チップで1Mビットのメモリが普通に使われ，4Mビットへの世代交代が進んでいます．もう，必要なメモリ空間をすべて実メモリとすることも，それほど困難ではありません．とすると，仮想記憶というのは今後は使われない技術なのでしょうか？　答はノーです．

仮想アドレスを物理アドレスに変換する2つの方式

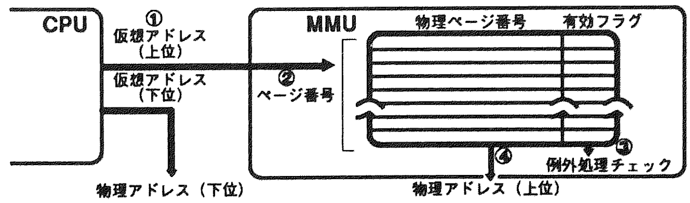
仮想記憶を用いたシステムでは，CPUから出力されるアドレスは仮想アドレスと呼ばれます．この仮想アドレスは，MMU (Memory Management Unit) を通して物理アドレスに変換されます．この変換の方法は，セグメンテーションとページングの2つの方式に大別できます．

セグメンテーション方式では，まずCPUから見たメモリ空間をインストラクション，データ，スタックなどに分割して管理します（この分割した単位がセグメント）．管理するために必須となるセグメントごとの情報は，そのセグメントが存在する物理空間での先頭アドレスとセグメントの大きさの2つです．実際のアドレスの変換方法はCPUに依存しますが，簡単なモデルだと以下のようになります．

- CPUから出力された仮想アドレスを上位2ビット（セグメントID）と，それ以外（セグメント内オフセット）に分けます．*1
- セグメントIDによって物理アドレスの先頭アドレスとセグメントの大きさを得ます．
- セグメント内オフセットがセグメントの大きさより大きい場合は例外処理に移ります．*2
- CPUから出力された仮想アドレスに対応する物理アドレスは，セグメントの物理空間での先頭アドレス＋セグメント内オフセット．

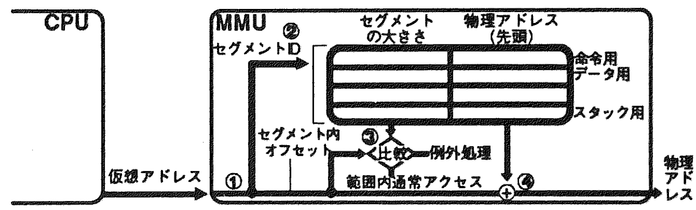
*1：セグメントIDというのは，一般的な用語ではありません．説明のためにつけたものです．
*2：スタックの場合には，通常アドレスは逆順ですから単純な比較ではダメで，少し工夫が必要です．

セグメンテーション方式



①仮想アドレスの上位2ビットをセグメントID，残りをセグメント内オフセットとする．
②セグメントIDをインデックスとしてセグメントの大きさと物理アドレス（先頭）を選択する．
③セグメント内オフセットとセグメントの大きさを比較し範囲チェックを行う．
④範囲チェックがOKのとき，物理アドレス(先頭)にセグメント内オフセットを加え，物理アドレスを出力．

ページング方式



①仮想アドレスの上位（20ビット）をページ番号とする．残りの下位12ビットはそのまま物理アドレスの下位部分となる．
②（仮想）ページ番号をページテーブルのインデックスとし，物理ページ番号と有効フラグを得る．
③有効フラグが「有効」と設定されていないときは例外処理．
④物理ページ番号を物理アドレスの上位として出力．

ページング方式では，メモリ空間はすべて固定のページと呼ばれる単位に分割します．ページの大きさは4KB程度です．ページの大きさは固定ですから，ページ内のオフセット（仮想アドレスの下位）はそのまま物理アドレスの下位アドレスになります．変換を行うために必須な管理情報は，ページごとに存在する物理ページ番号とそのページが有効かどうかを示すビットです．単純なモデルでは以下のように変換が行われます．

- CPUから出力された仮想アドレスをページ番号とページ内オフセットに分割します．
- ページ番号をインデックスとして変換テーブルの内容（物理ページ番号と有効／無効ビット）を得ます．
- ページの有効ビットが立っていない場合には例外処理に移ります．
- CPUが出力した仮想アドレスに対応する物理アドレスの上位アドレスは物理ページ番号，下位アドレスはページ内オフセット．

どちらの変換方式が有利？

セグメンテーション方式とページング方式のどちらが優れているかは，その目的などによっていちがいには判定できませんが，必要とするハードウェアが許容できる部品数でコスト面でも条件を満たすならば，ページング方式のほうが何かと有利です．以下にその

理由を述べます．

・管理する単位が固定の大きさなので，移動したり入れ替えたりする場合の処理が簡単で高速．

・管理する単位がページごとで比較的小さいのでより細かい制御が可能．

しかし，以下のような不利な点もあります．

・32ビットアドレスで4KB／ページ，ページごとに32ビットの情報が必要だとすると，ページの変換テーブルの大きさだけで4MB必要．

・セグメンテーション方式にくらべて必要な回路が複雑．

4MBだけですむかというとそうではなくて，複数のプロセスを管理することを考慮すれば，この数倍の変換テーブルが必要となってしまい，単純な方法では変換テーブルの大きさのほうが実メモリより大きいということになってしまいます．

ではどうするかというと，単純な一段の変換テーブルではなくて，複数の変換テーブル（2段から5段程度）で複数の変換を行うのが一般的です．

上の例では，32ビットのアドレスの上位20ビットを，直接変換テーブルのインデックスにしていました．これの20ビットを，たとえば5ビットずつに4分割すると，常に4MBの変換テーブルは必要ではなく，論理空間で使用している大きさに比例した大きさのテーブルだけが存在していればいいことになり，実際に計算すると最小限必要な変換テーブルの大きさは512バイトになります．

この場合でも使用する論理空間が大きくなると，全体で必要な変換テーブルの大きさは最大4MBになってしまいますが，ある特定の時点ではすべてのテーブルが存在している必要はなく，必要になった時点で設定をすればよいので，大きさ自体の問題は回避できます．

しかし，これらの変換テーブルをすべて1チップのMMUに内蔵したり，CPUに内蔵したりすることはできませんので，変換テーブルは通常のメモリ上に配置する必要があります．通常のメモリの変換テーブルが存在した場合には，そのままだと4段の変換テーブルの場合，1バイトのアクセスをするだけでも，メモリを5回アクセスしなくては変換できませんが，これでは実用になりません．

この問題の解決策は，CPU内部やMMUチップに変換キャッシュをもつことで解決できます．仮想アドレスに対する変換後のアドレスが，変換キャッシュに存在している場合にはそれを使用し，ない場合にはメモリ上の変換テーブルを使ってアクセスするのです．

どうですか？ 複雑ですよね．こんな複雑なことをするほどのメリットはあるんだろうか，という疑問があるでしょうが，最近のシステムでは，これくらいはごく普通の技術なのです．複雑になることによる部品点数の増加やコストの増加は，チップ自体の集積度が上がっていますから，これで十分クリアできます．変換を行うことによって，メモリへのアクセスが遅くなるという問題に関しては，たしかに初期のシステムでは仮想記憶を用いない場合にくらべて必ず遅くなっていました．

しかし，現在のシステムではいろいろな手法を用いることによって，仮想記憶のシステムでも変換時間を見かけ上ゼロにすることが可能なのでこれもとくに問題とはなりません．

なぜ，仮想記憶が必要か？

では，次に仮想記憶を用いると何が違うのかということを説明します．

・マルチタスク，マルチプロセス

仮想記憶を用いないシステムで複数のプロセスを動作させるためには，プログラムをすべてPC相対のアドレッシングモードを用いて書くか，プログラムの起動時にリンク処理（コンパイルの最後に行われるリンク処理と同じような処理でリロケーション情報の解決を行う）をする必要があります．

プログラムをすべてPC相対で記述するのは，特定のCPUでしかできませんし，一般的ではありません．プログラムの起動時のリンク処理は，起動までのスピードが遅くなります．

これらの問題が仮想記憶を用いることで解決できます．プロセスごとの空間は分離されているので，すべてのプロセスが同じ仮想アドレスを用いても，その物理空間は別々に割り当てることが可能だからです．

・プロセスの大きさの制限

仮想記憶を用いないシステムでは，実装されている実メモリより大きなプロセスを動作させることは不可能です．しかし，仮想記憶を用いたシステムでは，実メモリが不足した場合には，実メモリ上のデータをディスクに追い出し，また必要になったときに実メモ

りにもってくることによって，実メモリより大きなプロセスでも動作させることが可能です．

現在では，メモリチップあたりの容量が増えコストも下がっていますので，この目的だけで仮想記憶を採用することは少なくなってきました．しかし，通常は使わないようなプロセスもたくさんあり（たとえばプリンタスプーラなど），プログラムの中でも通常は使われない部分はたくさんあります．このように使われることが少ない部分のために実メモリを占有されなくてもよいというのはとても有用なことです．

・デマンドロード

仮想記憶を用いたシステムでは，プログラムを実行する場合にプログラムが格納されているディスクから，プログラムのすべてをメモリ上に必ずロードする必要があります．小さなプログラムだけならこれでもかまわないのですが，大きなプログラムになるとそのロードの時間だけで全体の処理時間の多くを占めてしまいます．

ページング方式ならば，このロード時間の多くの部分を節約することが可能です．どうするかというと，プログラムの必要な部分だけをロードしてすぐに実行してしまうのです．残りの部分は必要に応じてロードします．

これによって，使われない部分のロードの時間はすべて節約できますし，ディスクからDMAでロードしている最中にもプログラムを実行できます．

・プロセス間でのメモリの共有

複数のプロセスが簡単に動作させることが可能な環境では，同じプログラムを複数動作させることはごく一般的です．マルチユーザーとして使っている場合には，さらにその可能性が増えます．

その場合に，同じプログラムであれば命令コードなどのリードだけする領域は全く同じメモリ内容ですから，これらはひとつの物理メモリにおいて共有できます．これも仮想記憶のためのMMUがあれば簡単に実現できます．

また，ライトを行うデータ領域であっても，実際にライト操作が行われてからコピーを作成するという処理も可能で，システムによっては処理時間の短縮などにかなり有用な方法です．

・プロセス空間の保護

プログラムのバグやユーザーのまちがった操作によって暴走することがありますが，MMUがあればプロセスからアクセスできるメモリはそのプロセスのための空間だけに限定されますので，通常は暴走してリセットするしかないような事態のほとんどを回避することが可能です．

もっとも，回避したといってもそのプロセスを継続することはできませんし，メモリの管理そのものを行っているOSのバグの場合にはどうしようもありません．

4GB実装も夢ではない？

話はかわりますが，「もうそろそろ限界」といわれつつも，メモリチップの集積度って着実に上がっていますね．256Mビットでも時間の問題みたいです．

256Mビットというと，SIMM 1 本で256MBですから，SIMM 8 本で4GBになってしまい，32ビットCPUのすべての実メモリ空間にメモリを実装してしまうことも可能になります．「そんなにあっても全部使えるわけがない」という声が聞こえてきそうですが，そんなことはないと思います．

いままでは，CPUの処理速度，メモリ容量，ディスク容量で不可能だったことが可能になりますので，いま使っている使い方以外の用途が出てくるはずです．すぐに思いつくのは画像や音声などのイメージ処理です．

個人的に「これくらいの大きさのディスプレイが欲しいなあ」と思うのは，4096×4096の24ビットフルカラーですが，この場合ディスプレイ 1 枚分に必要なVRAMの大きさは48MBになります．

これらをマルチウィンドウで扱うとして，それに必要なメモリ空間は 8 倍だとして約400MB．これくらいのデータを扱うとすれば4GBのメモリがあっても，わりと簡単に全部使えてしまうんじゃないかなと思います．

もっともこの場合，現在のCPUの処理速度を考えるとメモリ容量とかではなくて，MIPS値のほうが先にネックになるでしょうね．静止画だけではなくて動画も（リアルタイムで計算しながら）扱いたいとなるとなおさらです．

あとどれだけかかるかはわかりませんが，非ノイマン型の並列処理CPUが一般的に使える状況になれば，たんなる夢の話とはならないでしょうし，早くそのような時代がくることを望んでいます．10年……じゃ無理かな．10年前といまをくらべてみて類推すると20年はかかりそうな気がします．

98活用アセンブラ技法

ビデオシステム編 第5回

バックカラー処理

星野 操

前回までに「構造化マクロ」「マクロユーティリティ」「外字を利用した半角文字処理」「常駐プログラムの技術」などを紹介し，これらを用いて，最終的に常駐タイプの枠線描画プログラムを作成しました．今回から，次のテーマとしてバックカラー処理をはじめることにします．

文字表示のアトリビュート

IBM PCとPC-9801との違い

IBM PCなどのシステムでは，文字を表示する場合に，バックカラーとフォアカラーを独立して指定することができます．IBMでよく使用されるアトリビュートに，暗い青バックに明るい白抜き文字，というものがあります．これは，なかなか画面に映えるものです．ただし，1文字に対するアトリビュートは8ビットしかありませんから，バックカラー8色，フォアカラー16色を指定するとなると，あと1ビットしか余裕がありません．このため，色以外のアトリビュートは，ブリンクしか指定することができません．

一方，PC-9801ではフォアカラー8色の指定しかできません．もっとも，アトリビュートで反転指定をすることができますから，バックカラーの選択をすることもできます．しかし，その場合はフォアカラーが黒になってしまいます．つまり，どちらか一方しか指定することができないのです．これは，PC-8801の時代のμPD3301というCRTチップのアトリビュートがそのようになっていたために，その流れを受け継いだPC-9801もそのようになったのでしょう．

PC-9801もIBM PC同様に8ビットのアトリビュートが指定できますが，その割り当ては色指定3ビット，アンダーライン，ブリンク，シークレット，バーチカルライン／簡易グラフィック，リバース指定各1ビットということになっています．バーチカルラインと簡易グラフィックは，ほかのハードウェアの設定によってその機能を選択します．

このうち，簡易グラフィックもやはりPC-8801の遺産であり，現実に使用されることはほとんどありませんから，事実上無意味です．シークレットにしても，色アトリビュートを指定できるようになっていれば，なくてもすむものです．バーチカルラインも，あまり使用されることはありません．アンダーラインは比較的有用だといえますが，全体にIBM PCに比べてあまり有用なアトリビュートが付加されているとはいいがたいものです．

PC-9801の場合，アトリビュートは物理的に一文字に1バイトしか割り当てられていません．しかし，1バイト空き領域があるので，そこにメモリを割り当てれば2バイトのアトリビュートを取ることができます．これを利用すれば，ハードウェアいかんでバックカラーの指定なども可能となるはずです．実際，PC-98Hではこの領域を利用して，フォアカラー／バックカラー双方を指定することが可能となっているようです．

PC-9801でのバックカラーの表示

PC-9801でも，バックカラーを出したいと思っていた人は少なくないようです．Turbo Cなどの画面でも，バックカラーを指定できるIBM版と，PC-9801版とではまったく「見映え」が違

います．また，Mifesなどでは，グラフィック画面に荒く青い点々を打ったり，アナログカラーパレットを操作して薄い青をバックに出して画面を見やすくするということが行われていました．

PC-9801でも，グラフィック画面を併用すればバックカラーを出すことは可能です．文字のバックは黒ではなく「透明」ですから，グラフィック画面の色がバックカラーとなるわけです．このため，文字を書き込むときに同時にグラフィック画面を操作すれば，バックカラーつきの表示が可能になるわけです．

ただし，半角文字でもグラフィック画面に最低16バイトの書き込みをする必要があり，ただキャラクタ画面に文字を書くよりは，ずいぶんスピードが遅くなってしまうことになります．CGなどのハードウェアを利用すれば16バイトの書き込みでよいのですが，もしなにもハードのサポートがないと，実に48バイトも書き込まなければいけません．GDCの塗り潰し機能を利用する手もあるのですが，あまり速度的には改善されません．今回は，対象をCG以上が搭載されているシステムとしているので，これを利用することにします．

バックカラー表示によるオーバーヘッド

さて，このようにバックカラーを出す場合には，少なからぬオーバーヘッドが必要になります．ですから，高速化にはかなり配慮しなければいけません．このためにいろいろなことが考えられます．

まずきわめて基本的なテクニックとして，「同一のバックカラーである場合にはグラフィック画面を重ね書きしない」というものがあります．非常に多くの場合，文字のフォアカラーが変わったとしても，バックカラーは変わらないものです．たとえばバックカラーが青のウィンドウの中をスクロールさせた場合，バックカラー自体はどこでも青のままです．ですから，わざわざ青に重ね書きをしなくても最初の1回だけ青にするだけで十分なのです．

さらに，機能の呼び出し方を工夫しなければいけません．1文字ごとの読み書きができることは必須といえます．しかし，この機能ユニットのみを利用することですべてをすませるということはできません．画面イメージを保存するために，2000回も読み取りのファンクションコールを発効するのでは，時間がかかってしょうがないからです．ですから，実際に画面を作成するうえにおいて，必要最小限の呼び出し形式を用意しなければいけません．

IBM PCの画面アクセスルーチン

PC-9801の場合は，画面の読み書きが比較的楽に行えることから，プログラムでは好きなところで自由奔放に画面の読み書きを行っていることが多いようです．しかし，IBM PCでは最初のグラフィックアダプタであるCGAがチラツキをおこす（昔のMZ-80Cのように）ため，水平・垂直帰線期間に読み書きをする必要がありました．ですから，同期を取って読み書きをするようなプログラムを書かなければいけなかったのです．

もちろん，最近のグラフィックアダプタは任意の時点で読み書きを行っても，ちらつきが生じるようなことはありません．しかし，IBM PCのプログラマは古いアダプタについての配慮を怠ることはしませんから，新しいアダプタのための高速な（同期を取らない）ルーチンのほかに，CGA用に同期を取って読み書きするルーチンを用意しています．

このように，単に文字を読み書きするだけでも非常に手間がかかったのです．ですから文字を読み書きするルーチンは，その典型的なものを用意して，それをプログラムの複数か所で呼び出すようにしてあります．1文字の読み書きだけでは呼び出しのオーバーヘッドが大きくなります．ですから，複数文字を一括して読み書きするようなルーチンも用意してあるわけです．

今回のような目的でも同様に，「典型的な」画面アクセスルーチンを用意する必要があります．IBM PCで用意されているもののなかには，具体的には次のようなものがあります．

画面の1文字の読み取り
画面への1文字の書き込み

これらは，とくにレジスタの内容と画面とのやり取りをするために便利です．複数の文字の読み書きルーチンで1文字だけ読み書きすることも可能ですが，その場合はオーバーヘッドが大きくなりますし，メモリバッファを用意する必要が生じてきます．

画面の複数文字の読み取り
画面の複数文字の書き込み

これらは，いわばrep movswのかわりに使用するものです．画面とのやり取りに必ずメモリバッファを使用する必要がありますが，画面イメージを保存したり復元したりするような場合に有効です．この場合には，メモリ内のイメージは文字コードとアトリビュートが混在しているものとなります．

画面への複数文字の埋め込み

rep stoswのかわりに使用するもの

です．おもに画面をクリアするために使用します．

画面間の文字の移動

画面のスクロールなどをする場合に必要となります．

固定アトリビュートの文字列の書き込み

アトリビュートを決めて文字列を書き込むもので，非常にひんぱんに使用されるものです．文字列を表示する場合，その1文字ごとにアトリビュートを指定することはあまりありません．1文字ぐらい変わる場合であれば，一括して表示したあとでアトリビュートだけ書きなおすということで対応できます．

この場合，アトリビュートはレジスタで指定でき，メモリ内のイメージはキャラクタのものだけに限定することができます．たとえば，Cの文字列定数をそのまま画面表示のソースとして使用できますから便利なわけです．もし，複数文字書き込みの機能を利用しようとすると，あらかじめプログラム側でアトリビュート指定を含んだ形に変更してやらなければいけません．

アトリビュートを変更しない文字列の書き込み

固定アトリビュートの文字列書き込みとほぼ同様ですが，特定アトリビュートを埋め込むことなく，文字列だけを書き込むものです．バックカラーがすでに設定されているウィンドウの中に文字列を書き込むような場合に利用すると便利です．

特定範囲へのアトリビュートの設定
特定範囲への文字コードの設定

文字コード，またはアトリビュートを変更することなく，他方のみに特定のコードを埋め込むものです．たとえば，一定範囲のアトリビュートを変更するとき，あるいは行末までの文字を消去するときなどに使用することができます．

文字コードの問題

さて，IBM PCの場合は，文字コードとアトリビュートがマージされた形になっています．半角1文字（IBM PCには全角はないけれど）は2バイトで構成され，下位バイトが文字コードを，上位バイトがアトリビュートを示しています．ですから，たとえば画面をクリアする場合には，空白コード20Hと通常白色アトリビュート07Hをあわせたワード0720Hをrep stoswのような形で画面に埋め込んでいけばよいことになります．

しかし，PC-9801の場合はそういうわけにはいきません．ハードウェア的には，半角1文字分のエリアに，2バイト分のキャラクタ領域が用意されています．それに1バイトのアトリビュート領域があるのです．ですから，普通に考えると3バイト，さらにバックカラーを拡張アトリビュートとして指定できるようにと考えれば，4バイトの領域が必要となります．

SJISコードで表現されている文字列に限定すれば，キャラクタは1バイト割り当てでもよくなります．ちょうどAXマシンのようになるわけです．ただし，その場合，画面上にSJISコードで表現できない文字が存在した場合には，とくにそれを読み込んだ場合に不都合が生じます．具体的には，半角のグラフィックキャラクタ，半角外字キャラクタ，半角罫線キャラクタなどが存在すると困ったことになるわけです．それらをバイト文字に変換したときに，別の文字として表現されてしまうからです．ですから，画面の退避・復元などの場合には文字コード2バイト，アトリビュート2バイトを単位としたバッファを対象にするようにしないといけません．

しかし，全部をその拡張されたコードで割り当てればよいかというと，そうもいきません．アトリビュート固定の文字列表示のような機能を呼び出す場合，プログラム中のおそらくはSJISコードで表現されているバッファの内容をいちいちJIS文字に変換して画面表示をさせるというのは不都合です．このため，とくに画面に書き込むためのファンクションはSJISコードそのままを取り扱うことができる必要があります．

また，この場合アトリビュートも1バイトとして扱えたほうが便利です．先に述べたように，PC-9801のアトリビュートにはあまり有効なものはありません．仮にフォアカラー・バックカラー各8色を使用するとして，余りのビットにブリンクとアンダーラインを割り当てれば，画面を構成するうえでさほどの不便は感じられなくなります．

よって，IBM PCで典型化されたファンクションのそれぞれについて，SJISコードとして（そしてアトリビュートは1バイトにエンコードされた形で）文字を取り扱うか，それぞれ2バイトの実際に画面表示に使われるJISコードで取り扱うかを検討しなければいけません．

プログラムの作成と注意点

1文字読み書きルーチンの作成

今回は，いちばん基本的な1文字読み書きのルーチンを作成し，それを作

成するうえでの注意点について書いていくことにします．

今回のプログラムbackcol. asmは，これで完結しているわけではありません．end疑似命令の前に，次回以降のプログラムを挿入することで機能を付加していくことにします．個別のモジュールにしてもよいのですが，その場合には各手続きをpublicにすることが必要です．

実行速度の改善

さて，今までのプログラムとは違って，今回の画面アクセスのようなプログラムは，簡単な処理の繰り返しが多数行われるものです．ですから，その単位実行速度の多少が繰り返しによって大きく響いてきます．

そのため，プログラムのサイズなどよりも実行速度に気を使う必要があります．もちろん，よいアルゴリズムを使用した場合，速度とともにサイズも小さくなることが多いものです．速度を上げるためには，

①アルゴリズムを改良する
②レジスタの使用法を最適化する
③手続きへのレジスタ渡しを実行する
④無駄なレジスタのスタック退避をしないように注意する
⑤関数呼び出しを利用せずにマクロ展開を利用する
⑥テーブリングを利用する
⑦ジャンプの利用を避ける

などの方法があります．

①は別格として，②，③，④は速度とともにサイズの縮小も期待できます．ただし，その分どの手続きがどのレジスタを利用しているかなどをきちんと管理しないといけません．プログラム自体の信頼性や保守性について

```
;----------------------------
;バックカラーの表示
; backcol: manage back color
;
; $Log: RCS/backcol.asm $
; revision 1.2 TAN 91/03/11 12:37:47
; argument modified.
;
; revision 1.1 TAN 91/03/11 12:35:50
; Initial revision
;
;
;
;----------------------------

        dosseg
        .model  small,pascal

        masm51                          ; delete if masm

        include condx.inc
        include utils.inc

GRAPH_SEG       equ     0a800h          ; graphic plane segment address
TEXT_SEG        equ     0a000h
ATTR_SEG        equ     0a200h
VIDEO_PLANE     equ     0a6h            ; video read/write plane set reg.
CG_CNT  equ     7ch                     ; CG command register
CG_MASK equ     7eh                     ; CG mask register

        .data
        even
attr2nd db      2048 dup (?)

        .code

;----------------------------
; バックカラーの設定
; set_back: backcolor set routine
; argument:
;       AL:back color code
;       DI:graphic plane base address
; return:
;       none
; broken:
;       DX,AH

        .data
last_back       db      ?               ; flag to reset CG mask register

        .code
        even
func    set_back        <>
        ; check if CG mask regist setting required
        cmp     al,last_back
        @if     nz
                mov     last_back,al
                ; set mask code
                irp     x,<ah,dl,dh>
                        ror     al,1
                        sbb     x,x     ; 0 if bit is not set, 0ffh if set
                endm

                ; now, ah:blue, dl:red, dh:green plane mask
                ; set mask register to CG
                mov     al,88h          ; plane 4 disable
                out     CG_CNT,al
                irp     x,<ah,dl,dh,0>
                        mov     al,x
                        out     CG_MASK,al
                endm
        @endif

        ; graphic plane write
        mov     al,0                            ; set plane
        out     0a4h,al
        out     0a6h,al

        push    ds
        mov     ax,GRAPH_SEG
        mov     ds,ax

        mov     ax,80
        rept    16
                mov     [di],al
                add     di,ax
        endm
        sub     di,80*16
        pop     ds

```

は，低下すると考えてよいでしょう．この部分とのトレードオフをしていることになります．

⑤，⑥はもろにサイズとのトレードオフです．⑤は深いループの中などで使用するととくに効果があがります．当然のことながらサイズが大きくなるので，十分に効用があるところだけで使うほうがよいでしょう．

⑥も同様です．たとえば，Cのctype.hのように各種文字種を判別するときでも，フラグテーブルを用意していて，それに対するアクセスを利用すれば高速に判別を行うことができます．今回のような場合，具体的には1バイト目のコードがSJISの漢字1バイト目であるかどうかを判別するときに，それを示す256バイトのフラグ配列を取っておき，そこをインデックスアクセスすれば1回で判別を行うことができます．ただし，テーブル領域がやはりサイズを喰うこと，V30／8086などのCPUではテーブルアクセスは意外と時間がかかる（実効アドレスの計算が遅い）ことにも注意しなければいけません．今回は，このためのテーブルは使用していません．

⑦のジャンプの利用，つまり分岐を避けるというのも重要なことです．分岐はキューの再フェッチを要求するので，その分実行速度が低下します．分岐を使用しないようにうまくプログラムを書くことは，①にもつながります．前記の例では，81H～9FHであるか0E0H～0FCHであるかを調べるために，へたをすると4回の条件ジャンプを使用しなければいけなくなります．これは下の下です．また，特別な条件でだけ実行されるような部分は，@ifなどでインラインにブロックを作成せずに，そのときだけ条件ジャンプでよそに飛ばして，また戻ってくるように

```
        ret
endf


;----------------------------
 ;CGを不動作状態にする
 ; disable_cg: disable CG
 ; argument:
 ;        none
 ; return:
 ;        none
 ; broken:
 ;        AL

        even
func    disable_cg      <>
        mov     al,0
        out     CG_CNT,al
        mov     last_back,0ffh
        ret
endf


;----------------------------
 ;1バイト形式のアトリビュートを2バイトに直す
 ; decode_attr: convert byte attribute to word attribute
 ; argument:
 ;        AL:attribute
 ; return:
 ;        AX:word size attribute
 ; broken:
 ;        DX
 ;
 ; byte type attribute is
 ;  8  7  6  5  4  3  2  1
 ; FG FR FB UL BL BG BR BB
 ;
 ; word type attribute is
 ; 16 15 14 13 12 11 10  9    8  7  6  5  4  3  2  1
 ;  -  -  -  -  -  - BG BR BB   FG FR FB VL UL RV BL /ST

        even
func    decode_attr     <>
        mov     ah,al
        mov     dx,ax
        and     ax,07e0h                ; set back/for attribute
        and     dx,1008h                ; extract UL/BL bit
        shr     dx,1
        shr     dl,1                    ; attribute position set
        or      al,dh
        or      al,dl
        or      al,1                    ; clear secret flag
        ret
endf


;----------------------------
 ;2バイト形式のアトリビュートを1バイトに直す
 ; encode_attr: convert word attribute to byte attribute
 ; argument:
 ;        AX:word size attribute
 ; return:
 ;        AL:attribute
 ; broken:
 ;        DX

        even
func    encode_attr     <>
        mov     dx,ax
        and     ax,07e0h
        or      al,ah                   ; back/for attribute
        mov     dl,dh
        and     dx,0802h                ; extract UL/BL bit
        shl     dx,1
        shl     dl,1
        or      al,dh
        or      al,dl
        ret
endf

;----------------------------
 ;jisコードをsjisコードに変換
 ; jis_to_sjis: convert jis to sjis code
 ; argument:
 ;        AX:JIS code, AH:second byte. ex:code 224ah->ah=4a,al=22
 ; return:
 ;        AX:SJIS code, AH:second byte
 ; broken:
 ;        none

```

すると普通のときにキューが破壊されないので速度が上がります．

これは，とくに80286／386などの高機能CPUで顕著になるでしょう．ノーウェイトシステムの場合，2バイトの命令フェッチに4クロックかかります．一方，mov ax, bxのようなプリミティブな命令は2バイトで2クロックで処理されてしまいます．ですから，CPUのキューは空いている状況が多いわけです．これに比べて，80286では命令フェッチに必要なクロックが半分，80386ではさらに1回に4バイトフェッチできるわけですから，キューが埋まっている状況のほうが多いのです．このため，キューが破棄される影響は，これらのほうが高いと思われます．

以上のように，速度を上げると，いろいろ注意したり美しくないことをしたりしなければいけなくなります．はたしてそこが速度を必要としているかどうかについては，よく考えてプログラムを書かないといけません．

今回の1文字読み書きというのは，比較的使用頻度が低いといえます．なぜなら，一括して画面にアクセスする場合には，ほかのサービスを利用すると思われるからです．呼び出しオーバーヘッドも大きいので，逆にいえば速度のチューンナップの必要性は比較的低いといえます．ですからマクロなどを多用する必要はなく，レジスタ渡しのサブルーチンを利用することを考えるくらいで十分といえましょう．

attr2nd

プログラムは，最初に内部で使用するセグメント値などの定数が記述され，そのあとにattr2ndという配列が確保されています．先に書いたように，バックカラーが変更されることはあま

```
        even
func    jis_to_sjis     <>
        sub     al,21h
        shr     al,1
        @if     c
                add     ah,7dh-1fh
        @endif
        cmp     ah,7fh-1fh
        cmc
        adc     ah,1fh

        add     al,81h
        cmp     al,0a0h
        @if     nc
                add     al,40h
        @endif
        ret
endf

;---------------------------
; sjisコードをjisコードに変換
; sjis_to_jis: convert sjis to jis code
; argument:
;       AX:SJIS code, AH:second byte
; return:
;       AX:JIS code, AH:second byte
; broken:
;       none

        even
func    sjis_to_jis     <>
        sub     al,71h
        and     al,3fh
        stc
        rcl     al,1
        cmp     ah,7fh
        cmc
        sbb     ah,1fh
        cmp     ah,7fh
        @if     nc
                inc     ax
                sub     ah,7fh-21h
        @endif
        ret
endf

;---------------------------
; 画面消去
; clear_scr: clear and sync screen
; argument:
;       none
; return:
;       none
; broken:
;       all except si/di

func    clear_scr       <>, <di>
        ; first, clear text screen
        mov     ax,TEXT_SEG
        mov     es,ax
        sub     di,di
        mov     cx,80*25
        sub     ax,ax
        rep     stosw

        ; set default attribute
        mov     ax,ATTR_SEG
        mov     es,ax
        sub     di,di
        mov     cx,80*25
        mov     ax,0e1h
        rep     stosw

        ; set cg to clear graphic plane
        mov     al,88h
        out     CG_CNT,al
        mov     al,0
        rept    4
                out     CG_MASK,al
        endm

        ; clear graphic plane
        mov     ax,GRAPH_SEG
        mov     es,ax
        sub     di,di
        mov     cx,40*25*16
        sub     ax,ax
        rep     stosw
```


ワンラインNEWS
東芝はアクセス速度17ナノ秒という世界最速の4MビットDRAMを開発した．従来は開発レベルで45ナノ秒だった．CPUのキャッシュ向けに製品化する予定．

りありません．バックカラーが変わらない場合は，書きなおす必要がないのです．そのメリットを活かすために，バックカラーが変わっていないかどうかを調べるためのペナルティは低くないといけません．このattr2ndという配列は，バックカラーを記憶するための配列領域です．これと要求されているバックカラーとを比較することで，高速に判断することができるのです．

set_back

次に，実際にバックカラーを設定するset_backという関数が定義されています．引数はレジスタ渡しで，DIにグラフィックプレーン上のアドレス，ALにバックカラーを設定します．DX／AHが破壊されていることにも注意します．

この関数は，別にlast_backという変数を使用しています．これには，直前にCGに設定されたバックカラーを設定しておきます．同じバックカラーを続けて書き込む場合には，CGのマスクレジスタの設定は不要です．ですから，これを見て不要な場合は設定をスキップすることで高速化がはかれます．そのあとのdisable_cgではCGのアクセスを不可にしていますが，このときにlast_backの値も－1にクリアしています．

関数は，最初にeven疑似命令でワードアラインメントを取っています．これも少しでも高速化をはかるためのものです．この関数は多くの場所から呼び出されることが期待されるので，evenをしています．一方，最後のput_chr／get_chrのような呼び出し頻度の低いものは，とくにevenをつけてもつけなくても影響はないと考えられます．

関数の中では，最初にCGのマスク

```
        disable_cg <>

        ; clear sub attribute code
        push    ds
        pop     es
        mov     di,offset attr2nd
        mov     cx,80*25
        sub     ax,ax
        rep     stosb

        ret
endf


;------------------------------
;アトリビュートのセット
; set_scr: set reverse attribute to back color
; argument:
;       none
; return:
;       none
; broken:
;       all except si/di

func    set_scr <>,<si di>
        sub     si,si
        sub     di,di
        mov     bx,offset attr2nd
        mov     cx,80
        @begin
                ; set graphic back color
                mov     ax,0ff00h
                mov     dx,GRAPH_SEG+1000h
                mov     es,dx
                cmp     es:[di],ah
                cmc
                rcl     al,1
                mov     dx,GRAPH_SEG+800h
                mov     es,dx
                cmp     es:[di],ah
                cmc
                rcl     al,1
                mov     dx,GRAPH_SEG
                mov     es,dx
                cmp     es:[di],ah
                cmc
                rcl     al,1
                mov     [bx],al

                ; check reverse
                mov     dx,ATTR_SEG
                mov     es,dx
                test    byte ptr es:[si],4
                @if     nz
                        ; extract color
                        mov     al,es:[si]
                        mov     cl,5
                        shr     al,cl

                        ; set backcolor buffer
                        mov     [bx],al

                        ; set for color black
                        and     byte ptr es:[si],1bh

                        ; graphic plane set
                        set_back <>
                @endif

                ; advance pointer
                inc     si
                inc     si
                inc     di
                inc     bx

                ; check boundary
                dec     cx
                @if     z
                        mov     cx,80
                        add     di,80*15
                @endif

                cmp     si,80*25
        @until  nc
        disable_cg <>
        ret
endf
```

レジスタに値を設定しています．先に述べたように，last_backを参照して必要のない場合はこれをスキップします．設定する場合には，last_backに値を保存したあと，AH／DL／DH各レジスタにCGのマスクレジスタの値を設定します．これが0であれば対応するグラフィックプレーンはクリアされ，0FFHであればグラフィックプレーンは表示されることになります．

この0／0FFHを設定するのは，ALの対応するビットをキャリーにローテート命令でセットし，sbb命令を使用することで達成できます．キャリーがセットされていれば0FFH，されていなければ0がレジスタに入るのです．これも条件分岐を使わないためのテクニックです（TKS>大人）．

次に，CGにコマンドを発行したあと，レジスタの値を次々にCGに設定していきます．今回は，4番目のグラフィックプレーンは使用しないことにしてあります．ですから4回目には0を出力します（オミットしてもかまわない）．

それから，描画・表示画面レジスタを設定した後，実際にグラフィックプレーンに書き込みをします．16バイトの書き込みが必要ですが，ここではループを使用せずにマクロ展開を利用しています．セグメントレジスタも破壊可能なESを利用せずに，DSを使用しています．これは少しでも速度を上げるためのものです．DIの値は最後にもとの値に復元して実行を終了します．

disable_cg

次のdisable_cgは，連続した画面アクセスのあとにCGを不動作状態にするためのものです．同時に，last_backの値もクリアします．

```
;----------------------------
;1文字書き込み
; put_chr: put byte/word char/attr
; argument:
;       mode:1 for word, 0 for byte
;       adr:text vram offset
;       code:byte/word character
;       attr:byte/word attribute
; return:
;       none
; broken:
;       all except si/di

        .data
; for double byte characters
        even
last_code       dw      ?
last_adr        dw      ?
last_attr       dw      ?

        .code
func    put_chr <mode, adr, code, attr> <di>
        ; check byte operation
        cmp     mode,0
        @if     z,far
                mov     bx,adr
                mov     ax,last_adr
                sub     ax,bx
                inc     ax
                inc     ax
                @if     z
                        ; second byte get
                        mov     ax,attr
                        decode_attr <>
                        mov     attr,ax

                        mov     ax,last_code
                        mov     bx,code
                        mov     ah,bl
                        sjis_to_jis <>
                        sub     al,20h
                        mov     code,ax
                        or      code,8080h      ; set r/l flag

                        ; write last byte
                        mov     bx,TEXT_SEG
                        mov     es,bx
                        mov     bx,last_adr
                        mov     es:[bx],ax

                        ; set attribute
                        mov     ax,ATTR_SEG
                        mov     es,ax
                        mov     ax,last_attr
                        mov     es:[bx],al

                        ; check back color
                        shr     bx,1
                        cmp     attr2nd[bx],ah
                        @if     nz
                                ; modify back color
                                mov     attr2nd[bx],ah
                                mov     dh,ah

                                ; calculate graphic address
                                xchg    ax,bx
                                mov     bh,80
                                div     bh
                                mov     bl,ah
                                mul     bh
                                add     ax,ax
                                add     ax,ax
                                add     ax,ax
                                add     ax,ax
                                mov     bh,0
                                add     ax,bx
                                xchg    di,ax

                                ; set back
                                mov     al,dh
                                set_back <>
                        @endif

                        ; clear flag
                        mov     last_adr,-1
                @else
                ; check 2byte first code
                mov     ax,code
                rol     al,1
                add     al,0c0h
```

decode_attr

その次のdecode_attrも，ひんぱんに呼ばれることが期待されるルーチンで，これは1バイト形式のアトリビュートを，2バイト画面表示用の形式になおすためのものです．実際には，バイト形式の書き込みを要求された場合に，このルーチンを利用して引数と実際に設定するアトリビュートとの変換をするわけです．ここも，条件ジャンプを利用しないで，DXレジスタを補助に使うことでキューの破壊を避けています．

バイトタイプのアトリビュートは，ここに書いてあるような形になっています．フォアカラー／バックカラーはそのままマスクするだけで使用できます．アンダーライン／ブリンクはそのために位置が異なってくるので，DXレジスタを利用して適切な位置にシフトしてやります．最後に，シークレットしないために1をorして変換を終了します．

encode_attr

encode_attrは，その逆に2バイトタイプのそれを1バイト形式に変換するものです．このルーチンは，逆にバイト形式の読み出しを利用するときに使われます．行っている内容も，単にシフトと論理演算をしているだけです．これらの変換もテーブリングを使うと高速に行うことができます．たとえば，decode_attrでは256＊2バイトのテーブルを使用すれば1回で変換できます．しかし，速度とサイズとを比較すると，それだけのメリットがあるかどうかはわかりません．

jis_to_sjis／sjis_to_jis

その次に，SJISとJISとの相互変換

```
                        not     al
                        test    al,81h
                        @e_if   z
                                ; may abandone last byte, set new code
                                mov     last_adr,bx
                                mov     ax,code
                                mov     last_code,ax
                                mov     ax,attr
                                decode_attr <>
                                mov     last_attr,ax
                                ret
                        @else
                                ; normal single byte character
                                mov     last_adr,-1

                                ; expand attribute
                                mov     ax,attr
                                decode_attr <>
                                mov     attr,ax
                        @endif
                @endif

                ; write it
                mov     ax,TEXT_SEG
                mov     es,ax
                mov     bx,adr
                mov     ax,code
                mov     es:[bx],ax

                ; set attribute
                mov     ax,ATTR_SEG
                mov     es,ax
                mov     ax,attr
                mov     es:[bx],al

                ; check back color
                shr     bx,1
                cmp     attr2nd[bx],ah
                @if     nz
                        ; modify back color
                        mov     attr2nd[bx],ah
                        mov     dh,ah

                        ; calculate graphic address
                        xchg    ax,bx
                        mov     bh,80
                        div     bh
                        mov     bl,ah
                        mul     bh
                        add     ax,ax
                        add     ax,ax
                        add     ax,ax
                        add     ax,ax
                        mov     bh,0
                        add     ax,bx
                        xchg    di,ax

                        ; set back
                        mov     al,dh
                        set_back <>
                @endif

                disable_cg <>

                ret
        endf

        ;----------------------------
        ; 1文字読み込み
        ; get_chr: get byte/word char/attr
        ; argument:
        ;       mode:1 for word, 0 for byte
        ;       adr:text vram offset
        ; return:
        ;       AX/AL:code, BX/BL:attr
        ; broken:
        ;       all except si/di

        func    get_chr <mode, adr>
                ; get code/attribute
                mov     ax,TEXT_SEG
                mov     es,ax
                mov     bx,adr
                mov     dx,es:[bx]                      ; code

                mov     ax,ATTR_SEG
                mov     es,ax
                mov     al,es:[bx]
                shr     bx,1
                mov     ah,attr2nd[bx]                  ; attr
```

をするルーチンが用意されています．バイト形式で受け渡しする文字列はSJISであるのに対して，画面表示に使用するのはJISコード（に近い形式）ですので，このようなルーチンが必要になります．

これらの変換をするアルゴリズムは，アスキー出版の『MS-DOSプログラミングテクニック』にあります．筆者も最初は，そのCの記述をそのままアセンブラに直したものを使用していましたが，今回はその条件分岐をやや少なくするための細工をしてみました．

もっとも，日経Mixのアセンブラ会議では，以前にこのようなSJIS←→JIS変換ルーチンを作成する，という話題が出たことがあり，そのときには16バイト程度の非常に短い，高速なルーチンが作成されました．それらをそのままここに載せるのも何ですので，今回は前記のような，必ずしも効率はよくないルーチンを採用しました．もちろん，日経Mix上のルーチンに載せ変えれば，より高速になります．

これらのルーチンでは，キャリーを操作することによって分岐を多少避けるようにしています．たとえば，7FHと比較をしたあとcmcしてsbbすることで分岐を１つ減らしているわけです．分岐を減らすためには，キャリーをうまく使うことが必要です．

clear_scr

その次の，clear_scrというのは，画面を消去するためのルーチンです．そのあとのset_scrかどちらかを，このライブラリを使用する前に呼び出さないといけません．ライブラリが関与しないdos出力などが行われたあとも同様です．これらは，とくにattr2ndとの同期を取るために使用します．

```
        mov     bx,dx
        xchg    ax,bx

        cmp     mode,0
        @if     z
                ; byte type,  code check
                or      ah,ah
                @if     nz
                        add     al,20h
                        and     ax,7f7fh
                        jis_to_sjis <>
                        test    dx,8080h
                        @if     nz
                                ; right/2nd byte
                                mov     al,ah
                        @endif
                        mov     ah,0
                @endif

                ; encode attribute
                xchg    ax,bx
                encode_attr <>
                xchg    ax,bx
        @endif
        ret
endf

        end     start

```

このライブラリが正常に動作するためには，バックカラーアトリビュートが正常にattr2ndに格納されていることが必要です．DOS出力などがあるとそれが保証されない（反転文字を表示しなくても，スクロールされるだけで同期が取れなくなる）ためです．

最初のclear_scrは，単にテキストアトリビュートをクリアし，グラフィック画面もクリアし，attr2ndもクリアしてしまうだけです．このときにも，GCを利用することで１画面のみクリアすることですましています．DIレジスタを使用するので，func宣言のところで〈di〉と指定してあります．

set_scr

次のset_scrは，実際に画面をスキャンしてアトリビュートのセットなどを行うものです．まず，DIで示されるグラフィックプレーン上のポインタから各カラープレーンのデータを読み出し，これらが0FFHであればそのバックカラーがセットされていると判断します．そして，それらをattr2ndに設定します．ここで，エンコードもキャリーを利用して条件分岐をせずにすむようにしてあります．

それから，テキストのアトリビュートをチェックし，これのリバースビットが設定されているかどうかをチェックします．もしリバースであれば，アトリビュートに設定されている色をバックカラーとして処理し，フォアカラーは黒にします．そしてset_backを呼び出して実際にグラフィック画面を利用し，バックカラーを表示します．

これによって，テキストのバックカラー表示がグラフィックを利用したバックカラー表示にすりかえられ，かつその値がattr2ndにも設定されるわけです．

あとは，各ポインタをインクリメントします．CXはカラムカウンタで，80カラムを超えたときには，DIポインタの値を次の文字の上端にあたるグラフィックポジションに更正します．

以上の処理をすべてのキャラクタについて行い，最後にCGをディゼーブルして終了します．

put_char

さて，最後にこれらのサブルーチンを利用して読み書きを行う関数を示します．まず，put_charは1文字の書き込みをするルーチンです．これはもともと呼び出しのオーバーヘッドが大きい関数ですから，引数はスタック渡しにしてしまいます．これによって，高級言語から直接呼び出すこともできるようにしているわけです．

引数は先に述べたように，読み書きを物理システムに合致したワード単位で行うか，SJISコードであるバイト単位で行うかを示すモード，それからテキストプレーンでの書き込みアドレス，コード，アトリビュートとします．コード，アトリビュートはモードにしたがってワード／バイトが決定されます．

バイト単位の場合，それが漢字など2バイトコードの先頭バイトであれば記憶しておき，次の段階で書き込まないといけません．このための変数をデータセグメントにlast_code／last_attr／last_adrとして用意しておきます．

まず，バイト単位であれば次のコードが2バイト目であるかどうかをチェックします．ここでは，2バイト目のレンジチェックは行っていません．条件としては，アドレスが連続していることが要求されます．もしアドレスが連続していなければ，前のデータは破棄される可能性があります．2バイト目のコードが出力されれば，アトリビュートとデータを拡張します．ここで，decode_attrとsjis_to_jisが呼び出されます．画面表示コードでは，先頭バイトから20Hを引く必要があることに注意します．

左側の文字はそのままのコードを書き込みますが，右側の文字はフラグをセットしないといけません．このフラグは，実はハードウェア的にはまったく意味をもっていないものです．単にソフトウェア上の目印にすぎません．このビットの立て方もいろいろあるのですが，ここでは両方のバイトの最上位ビットをセットすることにします．さらに右であることを判定するには，いずれかのビットが立っていればよしとします．

右側に相当する部分はあとで書き込まれるので，先に前回の呼び出しで書き込まれるはずであった左側の文字を書き込みます．校正された文字コードをテキストセグメントに，用意されているアトリビュートの下位バイトをアトリビュートセグメントにセットします．さらに，BXを右に1ビットシフトし，attr2ndのオフセットに合わせてバックカラーのチェックをします．もしバックカラーが異なっていれば，それをエリアにセットしたあとグラフィック画面上のポインタを計算します．

これも配列を利用して計算する方法はあるのですが，簡単に，わり算／乗算を利用して文字のx／y位置を算出し，それをグラフィック位置に直します．あとはset_backを利用してバックカラーを設定するだけです．last_adrはこの後にクリアしておきます．

さて，2バイト目でない場合は，まず漢字の1バイト目であるかどうかをチェックします．ここでは簡単に80H～9FHと0E0H～0FFHは1バイト目であるとします．このレンジであることをチェックするには，先に述べたようにテーブルを利用する方法もありますが，今回はちょっとした細工を利用しています．つまり1バイト目である場合には，上位3ビットが100か111であるわけですから，左に1ビットローテートして40Hをたすと，11xxxxx1か10xxxxx1となります．ですから，この最上位／最下位ビットをチェックすればよいわけです．

1バイト目と判断されれば，引数をワークエリアに退避します．ここで，アトリビュートだけは事前に拡張したものをセットしておきます．そして，直接リターンしてしまいます．

いずれにもあてはまらなかった場合には，1バイトコードを出力するようになっているわけですから，last_adrをクリアし，アトリビュートを拡張します．

それから，ワードモードの場合にはワードデータを，バイトモードの場合は漢字の右コードか1バイト文字を書き込みます．このあたりの処理は漢字左文字を書き込む場合と同様です．そして，最後にCGをディゼーブルします．

get_char

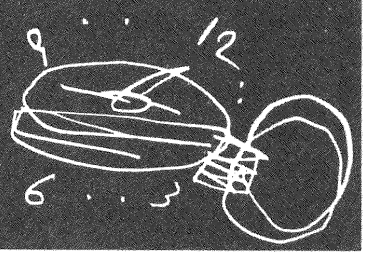
get_chrは読み込みを行うものです．やはり，引数はモードとオフセットアドレスを取ります．返り値は，AX／BXレジスタにワード／バイトいずれかの単位で返します．まず，アトリビュートとコードをワード形式で読み込みます．もしバイトモードであれば，2バイトコードなら20Hの補正をしてSJISに変換します．そして，フラグにしたがって上位または下位のバイトを取り出します．アトリビュートもエンコードします．値はAX／BXに返します．

次回は，さらに複数文字列の読み書き処理やアナログパレットを利用したバックカラーの設定，ライブラリの使用例などについて可能な範囲で触れることにします．

Illustration : Masami Nozue

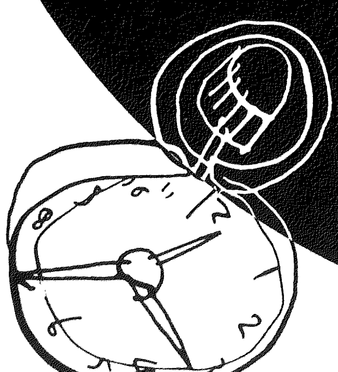
実践C塾 第5回

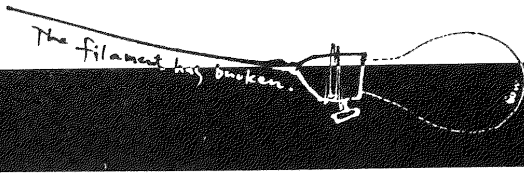
プロセス制御/システムコールライブラリ



塾頭 水居 浄

今回は，前回にひきつづき，Microsoft CとTurbo Cのランタイムライブラリにおける，処理系固有のライブラリ関数の比較検討を行います．とくに，システムコールライブラリは重要です．実際のアプリケーションの開発では，ANSI規格の範囲内の標準ライブラリではできないような細かい処理を行わなくてはならないことが非常に多いからです．





1 プロセス制御ライブラリ

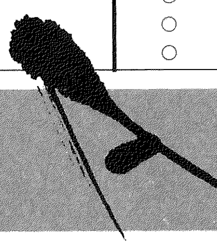
プロセス制御とは，メモリ上で実行されるプログラムを制御することです．もともと，マルチタスクシステムでなければあまり意味がありません．しかし，シングルタスクシステムであるMS-DOSでも，プログラムからほかのプログラムを起動して実行するということはできます．MS-DOSでは，呼び出した「子」のプログラムを実行している間は，「親」のプログラムは一時停止して「子」の終了を待つという形になるわけです．したがって，かなり制限された形ですが，MS-DOS用にもプロセス制御のためのライブラリ関数が用意されています．

表1にMicrosoft CとTurbo Cのプロセス制御ライブラリ（MS-DOSのみ）の各関数名を示します．

子プロセスを起動する関数には，exec系，spawn系およびsystemの3種類があります．

表1　プロセス制御ライブラリ

	MSC6.0	MSC5.1	TC++	TC2.0	互換性
abort	○	○	○	○	ANSI
atexit	○	○	○	○	ANSI
exec系	○	○	○	○	UNIX
exit	○	○	○	○	ANSI
_exit	○	○	○	○	UNIX
getpid	○	○	○	○	UNIX
onexit	○	○	—	—	
raise	○	○	○	○	ANSI
signal	○	○	○	○	ANSI
spawn系	○	○	○	○	
system	○	○	○	○	ANSI



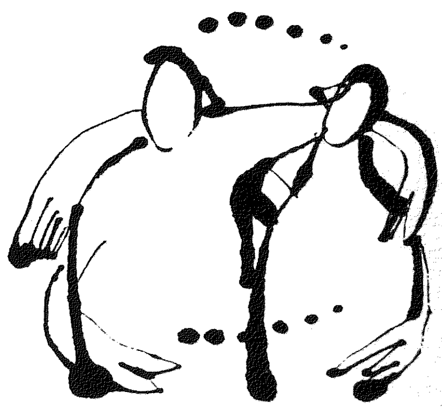
spawn系の関数
親プロセスは，子プロセスを起動した時点で一時停止

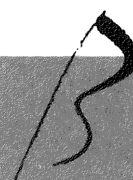
spawn系の関数には，spawnl, spawnle, spawnlp, spawnlpe, spawnv, spawnve, spawnvp, spawnvpeの8種類があります．これらは，末尾の文字によって細かい意味が異なります．spawnlとspawnvは，コマンドライン引数の与えかたの違いです．spawnlでは，コマンドライン引数をこの関数の引数として並べて記述するのに対して，spawnvでは，コマンドライン引数をmainのargvと同様のポインタ配列に格納して与えます．プログラムの組み方によって，どちらが便利かは変わってきますので，2通り用意してあるわけです．

spawnle, spawnveは，環境を指定して引き継がせる指定です．通常，子プロセスは親プロセスの環境をそのまま引き継ぎますが，それとは異なる環境を引き継がせたい場合に，これらの関数の引数envpを使って指定します．

spawnlp, spawnvpは，子プロセスをMS-DOSのコマンドサーチパス上で探して起動させる指定です．spawnlpe, spawnvpeは環境の指定とパスサーチを両方行う指定になるわけです．

spawn系関数の場合，親プロセスは子プロセスを起動した時点で一時停止して，子プロセスの終了を待ちます．子プロセスが終了すると，この関数呼び出しから復帰するわけです．関数の戻り値は，子プロセスの戻り値になります．





exec系の関数
親プロセスから子プロセスへバトンタッチ

exec系の関数にも，execl, execle, execlp, execlpe, execv, execve, execvp, execvpeの8種類があります．これらの種類の意味はspawn系関数と同様ですが，exec系の関数は子プロセスの起動に成功したら，親プロセスには戻ってきません．子プロセスを起動したところで，親プロセスをメモリ上から除去してしまいます．すなわち，プログラムから次のプログラムへバトンタッチして処理を続ける形になるわけです．

system関数
COMMAND.COMを起動してから子プロセスを起動

system関数はCOMMAND.COMを起動して，さらにそのコマンドライン上から子プロセスを起動するものです(COMMAND.COMの/Cオプションを使います)．この方法の利点は，リダイレクトやパイプなどの処理を容易に指定できるということです．逆に欠点としては，COMMAND.COMを経由するために処理が若干重くなることと，子プロセスの返す戻り値を親プロセスが取得できないという点です．したがって，子プロセスの返した戻り値によって処理を変えたいといった場合には使いにくくなります．

UNIXのfork
プロセスの複製を作って起動

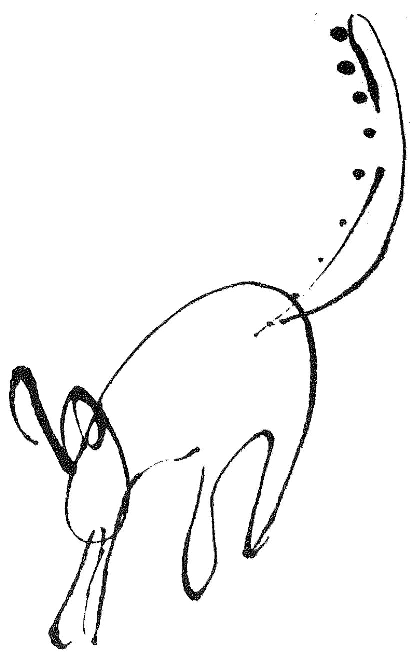
これらのプロセス制御関数のモデルとなったのはUNIXのライブラリ関数ですが，UNIXにはspawn系の関数はありません．そのかわりにforkという関数を使用します．この関数は，実行したプロセスの複製を作って起動するというものです．すなわち，あるプログラムがforkを呼び出した時点で，そのプログラムの複製がメモリ上に作られ，子プロセスとして動作を開始するわけです．forkの戻り値によって，親と子は区別されます．一般には，子のほうではただちにexec系関数を使って別なプログラムを起動し，spawn系関数と同等の処理を行うわけです．このfork + execよりは，明らかにspawnのほうがすっきりしていますね．spawn のほうが無駄が少なく，処理も軽いはずです．

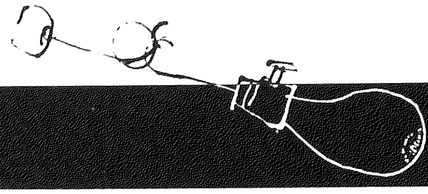
何のためにUNIXではこういう仕組みになっているのかというと，やはりマルチタスクシステムの利点を生かすためだと思います．子プロセスのほうでexecを呼び出さずに，親プロセスと同じプログラムを「その時点」から並行動作させることができますから，たとえば，将棋のようなゲームのプログラムで，複数のプロセスで並行して別々のアルゴリズムを試してみて，よりよいほうを採用するといったプログラムを簡単に記述できます．MS-DOSはマルチタスクシステムではありませんから，forkを採用する意味がないという判断からサポートされていないようです．

Microsoft Cの spawn & system
環境変数に;C_FILE_INFOを追加

Microsoft C 5.1では，spawnやsystemで子プロセスを起動すると，環境変数に;C_FILE_INFOという変なものが追加されます．これは，親プロセスでオープンしていたファイルに関する情報を子プロセスに引き継がせるためのものです．一般にオープンしていたファイルの情報は，何もこんなものを使わなくてもMS-DOSが自動的に引き継がせますから，このようなものは不要に思えます．しかし，Cのランタイムライブラリでは，ファイルについてテキストモードとバイナリモードという2つのモードを定義しています．この「モード」はMS-DOSが関知しないものですから，これに関する情報まで引き継がせようとした場合は，こういった方法を使う必要があるのです．

もっとも，;C_FILE_INFOで渡された情報は子プロセスのスタートアップルーチンで処理されますから，子プロセス側もMicrosoft Cで記述していないと意味がありません．それに実際には，こういう情報の引き渡しまで必要とするケースはそれほど多くありませんし，子プロセスとしてCOMMAND.COMを起動したときに，setコマンドで;C_FILE_INFOが見えてしまうのはあまり美しくないという話もあります．そこで，Microsoft C 6.0では，stdlib.hで宣言されている_fileinfoという外部変数の値が0以外のときだけ，環境変数を使ってファイルの情報を引き渡すというように仕様が変更されています(ちなみに，環境変数名も_C_FILE_INFOに変更されています)．なお，Turbo Cにはこういう機構自体が存在しないようです．





2 システムコールライブラリ

ANSI規格準拠のライブラリだけで，MS-DOS上の高機能なアプリケーションを作成するのはかなり無理があります．画面やキーボードのきめ細かな制御は，ANSIライブラリだけではとうてい不可能に近いといえます．こういった要求を満たすためには，MS-DOSのシステムコールや，BIOSのシステムコール，さらにはVRAMなどの直接操作といった手法によるライブラリを構築しなくてはなりません．こういう用途のため，システムコールを直接行うライブラリが両処理系にはサポートされています．

表2の通り，両処理系間では，ほぼ完璧に互換性がありません．同じ機能をサポートしている関数は多いのですが，ほとんどすべて関数名や細かい仕様が異なっています．もともとこの分野のライブ

表2　システムコールライブラリ

	MSC6.0	MSC5.1	TC++	TC2.0	互換性
absread	—	—	○	○	
abswrite	—	—	○	○	
allocmem	—	—	○	○	
bdos	○	○	○	○	Lattice
bdosptr	—	—	○	○	
bios系	—	—	○	○	
bios98系	—	—	○	○	
_bios系	○	—	—	—	
_chain_intr	○	○	—	—	
_chdrive	○	—	—	—	
_chmod	—	—	○	○	
_close	—	—	○	○	
country	—	—	○	○	
_creat	—	—	○	○	
creatnew	—	—	○	○	
creattemp	—	—	○	○	
ctrlbrk	—	—	○	○	
disable	—	—	○	○	
_disable	○	○	—	—	
_dos_allocmem	○	○	—	—	
_dos_close	○	○	—	—	
_dos_creat	○	○	—	—	
_dos_creatnew	○	○	—	—	
dosexterr	○	○	○	○	
_dos_findfirst	○	○	—	—	
_dos_findnext	○	○	—	—	
_dos_freemem	○	○	—	—	
_dos_getdate	○	○	—	—	
_dos_getdiskfree	○	○	—	—	
_dos_getdrive	○	○	—	—	
_dos_getfileattr	○	○	—	—	
_dos_getftime	○	○	—	—	
_dos_gettime	○	○	—	—	
_dos_getvect	○	○	—	—	
_dos_keep	○	○	—	—	
_dos_open	○	○	—	—	
_dos_read	○	○	—	—	
_dos_setblock	○	○	—	—	
_dos_setdate	○	○	—	—	
_dos_setdrive	○	○	—	—	
_dos_setfileattr	○	○	—	—	
_dos_setftime	○	○	—	—	
_dos_settime	○	○	—	—	
_dos_setvect	○	○	—	—	
_dos_write	○	○	—	—	
enable	—	—	○	○	
_enable	○	○	—	—	
findfirst	—	—	○	○	
findnext	—	—	○	○	
freemem	—	—	○	○	
geninterrupt	—	—	○	○	
getcbrk	—	—	○	○	
getcurdir	—	—	○	○	
getdate	—	—	○	○	
_getdcwd	○	—	—	—	
getdfree	—	—	○	○	
getdisk	—	—	○	○	
_getdrive	○	—	—	—	
getdta	—	—	○	○	
getfat	—	—	○	○	
getfatd	—	—	○	○	
getpsp	—	—	○	○	
gettime	—	—	○	○	
getvect	—	—	○	○	
getverify	—	—	○	○	
harderr	—	—	○	○	
_harderr	○	○	—	—	
hardresume	—	—	○	○	
_hardresume	○	○	—	—	
hardretn	—	—	○	○	
_hardretn	○	○	—	—	
inp	○	○	—	—	Lattice
inport	—	—	○	○	
inportb	—	—	○	○	
inpw	○	○	—	—	
int86	○	○	○	○	Lattice
int86x	○	○	○	○	Lattice
intdos	○	○	○	○	Lattice
intdosx	○	○	○	○	Lattice
intr	—	—	○	○	
keep	—	—	○	○	
_open	—	—	○	○	
outp	○	○	—	—	Lattice
outport	—	—	○	○	
outportb	—	—	○	○	
outpw	○	○	—	—	
parsfnm	—	—	○	○	
peek	—	—	○	○	
peekb	—	—	○	○	
poke	—	—	○	○	
pokeb	—	—	○	○	
randbrd	—	—	○	○	
randbwr	—	—	○	○	
_read	—	—	○	○	
segread	○	○	○	○	Lattice
setblock	—	—	○	○	
setcbrk	—	—	○	○	
setdate	—	—	○	○	
setdisk	—	—	○	○	
setdta	—	—	○	○	
settime	—	—	○	○	
setvect	—	—	○	○	
setverify	—	—	○	○	
sleep	—	—	○	○	
_write	—	—	○	○	

ラリは，Microsoft CではV.5.0以前は大部分がサポートされておらず，int86などのLattice系のライブラリしかありませんでした．

一方，Turbo CではV.1.0からこの分野をサポートしていました．Microsoft CがV.5.1でこれらのライブラリを追加したとき，Turbo Cとの互換をあえて取らなかったため，このような非互換性が生じてしまったといえます．互換を取らなかったのは，1つには著作権関係の懸念もあったのでしょうが，関数のネーミングという点でもTurbo Cの仕様はあまりよくないという判断が働いたようです．たとえばkeepという関数は，TSRを作成するときに使用する常駐終了のための関数ですが，ユーザーが他の用途にそういう関数名を使いたいという状況も容易に予想できます．ユーザーが使う可能性の高いこういう一般的な名前を，特殊なライブラリ関数名に割り当ててしまうのはあまり好ましいこととはいえません．Microsoft Cでは，関数名の先頭に_dos_をつけるようにして，この問題を回避しています．

int86系関数
高級言語から直接システムコールを呼び出す

MS-DOSやBIOSのシステムコールの呼び出しには，通常，ソフトウェア割り込み命令（INT xx）を使用します．これはレジスタでパラメータを渡すアセンブリ言語用のインターフェースであるため，一般にC言語からそのまま利用することはできません．Microsoft CやTurbo Cでは，ランタイムライブラリの下位ルーチンをアセンブリ言語で記述してこれらのコールを行っていますが，ユーザーのプログラムから直接呼び出したい場合に，アセンブラを併用しなくてはならないというのでは不便です．そこで，Microsoft CやTurbo Cでは，各レジスタに変数の値を設定してINT命令を発行するサービスを行うint86，intdosなどといった関数をサポートしています．これらの関数は，もともとはLattice Cが最初にサポートしたものをMicrosoft CやTurbo Cが継承し，少しずつ機能拡張したものです．

int86とint86xは，割り込み番号を指定して，任意のソフトウェア割り込みを実行することができるものです．一般に，機種依存したBIOSなどの呼び出しに使用します．bdosとintdos，intdosxは，MS-DOSのINT 21hによるシステムコール専用のものです．

int86やintdosでは，ソフトウェア割り込みを実行する際に各レジスタに与える各種パラメータを，REGS構造体を使って指定します．この構造体はリスト1のようなものです．

各メンバ名は各レジスタをそのまま表しています．共用体とすることで，AX～DXレジスタをバイト単位でも操作できるようにしています．cflagというのはキャリーフラグのことで，INT命令実行直後のキャリーフラグの状態を保持します．これは，一般にシステムコールがエラーの有無をキャリーフラグの状態を使って返すものが多いためです．

flagsというのは，キャリーも含めた全フラグの状態を保持するものです．これはMicrosoft Cにはなく，Turbo Cが拡張したものです．INT 21hファンクションコールでは，キャリー以外のフラグで状態を返すものはほとんどありませんから，cflagだけでも問題にはなりませんが，BIOSなどではほかのフラグを使う可能性がありますので，Microsoft Cの仕様では少々機能不足であるといえます．

具体的な使用例として，INT 21hファンクション36hをintdosを使って呼び出すことにより，ディスクの空き容量をバイト単位で取得する関数を作った場合をリスト2に示します．

int86xやintdosxでは，REGS構造体に加えてSREGS構造体を使ってセグメントレジスタの値を指定します．この構造体はリスト3のようなものです．

リスト1　REGS構造体

```
struct WORDREGS {
    unsigned int ax, bx, cx, dx, si, di, cflag, flags;
};
struct BYTEREGS {
    unsigned char al, ah, bl, bh, cl, ch, dl, dh;
};
union REGS {
    struct WORDREGS x;
    struct BYTEREGS h;
};
```

リスト2　ディスクの空き容量をバイト単位で取得する関数

```
long GetDiskFree(short nDrive)
{
    union REGS inregs, outregs;

    inregs.h.ah = 0x36;
    inregs.h.dl = nDrive;
    intdos(&inregs, &outregs);
    if (outregs.x.ax == 0xFFFF)
        return -1L; /* error */
    return (long)outregs.x.bx * outregs.x.ax * outregs.x.cx;
}
```

セグメントレジスタにもパラメータを指定する必要があるシステムコールでは，int86xやintdosxを使用しなくてはなりません．

具体例として，INT 21hファンクション56hをintdosxを使って呼び出すことにより，ファイル名を変更する関数を作った場合をリスト4に示します（renameというライブラリ関数がすでにありますから，わざわざ自作する必要はないのですが，あくまで例として提示します）．

先ほどのintdosの例と違って，REGS構造体を入力側と出力側で共用させてしまっている点に注意してください．一般にこれは共用が可能で，入力側の構造体に与えた値を保存する必要がないなら，とくに分ける必要はありません．SREGS構造体については，入力側と出力側が常に共用されています．すなわち，セグメントレジスタに何か値を返してくるようなシステムコールでは，SREGS構造体に値が設定されます．なんとも統一性のない仕様ですが，そうなっている以上しかたがありません．

なお，SREGS構造体には4つのセグメントレジスタがすべてメンバとして宣言されていますが，実際にユーザーが設定できるのは，DSとESの2つだけです．CSとSSについては値を設定しても無視されます．

PC-9801のBIOSを使ったことがある方ならすぐに気がつくと思いますが，REGS構造体にはBPレジスタがありません．これは，PC-9801のディスクBIOSのように，BPレジスタを使ってパラメータを渡すようなシステムコールには，int86系関数は使用できないということを意味します．もっとも，BPレジスタは高級言語でフレームポインタとして使われることを主要な目的としているものですから，BIOSのパラメータとして使うという用途はかなり特殊であるといえます．

リスト3　SREGS構造体

```
struct SREGS {
    unsigned int es,cs,ss,ds;
};
```

リスト4　intdosxを使用したファイル名変更関数

```
int RenameFile(char *pszOrgName, char *pszNewName)
{
    union REGS regs;
    struct SREGS sregs;

    sregs.ds  = FP_SEG(pszOrgName);
    regs.x.dx = FP_OFF(pszOrgName);
    sregs.es  = FP_SEG(pszNewName);
    regs.x.di = FP_OFF(pszNewName);
    intdosx(&regs, &regs, &sregs);
    if (regs.x.cflag != 0)
        return -1; /* error   */
    return 0;      /* success */
}
```

リスト5　REGPACK構造体

```
struct REGPACK {
    unsigned r_ax, r_bx, r_cx, r_dx;
    unsigned r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
```

実際，IBM PCやFMRなどではBPをパラメータに使うBIOSはありません．したがって，これはint86系関数の仕様の不備というより，PC-9801のBIOSの仕様のほうが非常識であるといえます．

このBPレジスタの問題は，Microsoft Cではアセンブラを使う以外に回避方法がないのですが，Turbo Cの場合はintrという関数を使う方法があります．このintrは，リスト5のようなREGPACK構造体を使用するので，BPレジスタを扱うことができます．

仕様としては，int86xよりもintrのほうがスッキリしているのですが，Turbo Cにしかないので移植性は低くなります．

ところで，int86やint86xでINT 21hを呼び出すことももちろん可能です．では，何のためにintdosやintdosxがあるのでしょうか？　実は，これは内部的な効率の問題によるのです．8086のアセンブリ言語の経験がある方はご存じだと思いますが，INT命令で指定する割り込み番号には定数値しか許されていません．すなわち，どこか適当なデータセグメント中に番号を記録して，それを参照して割り込みをかけるということはできないのです．したがって，int86のような汎用の関数を実現するには，自分自身のコードを動的に書き換えるようなことをしないといけないわけです．実際には，int86やint86xでは次のような短いプログラムを一時的にスタック上に作成して，それをfarコールするという方法で実現しています．

```
INT     ベクタ番号
RETF
```

もちろん，intdosやintdosxでは，割り込み番号は固定（21h）ですから，こういうラジカルな方法を使う必要はありません．したがって，INT 21hを呼び出すときには，int86を使うよりintdosを使うほうが効率がよくなるのです．

INT 21hを呼び出す関数としては，ほかにbdosというものがあります．これ

は，INT 21hの中でも比較的機能が単純でAX，DXレジスタだけしか使わないファンクションに限って使用できる関数です．REGS，SREGS構造体を使わないため，intdosより効率がよくなります．Turbo Cでは，これに加えてbdosptrという関数もあります．これは，AX，DXに加えて，DSも扱えるようにしたものです．



MS-DOSシステムコール関数
intdosの無駄をはぶくライブラリ

MS-DOSのシステムコールは，上記のintdosなどで一応全部カバーできますが，各ファンクションコールはREGS構造体にあるレジスタを常に全部使うわけではありませんから，intdosでは何かと無駄が多くなります．また，いちいち構造体に値をセットしたり，構造体から値を取得したりする手間もめんどうなものです．そこで，Microsoft CやTurbo Cでは，よく使うシステムコールについては個別のライブラリを用意しています．

表３にMS-DOSシステムコールに対応した各ライブラリ関数を示します．

Turbo Cのほうが若干守備範囲が広くなっています．Microsoft CとTurbo Cで互換性があるのはdosexterrだけですね．ほかは関数名が異なりますし，引数そのほかの細かい仕様も異なります．

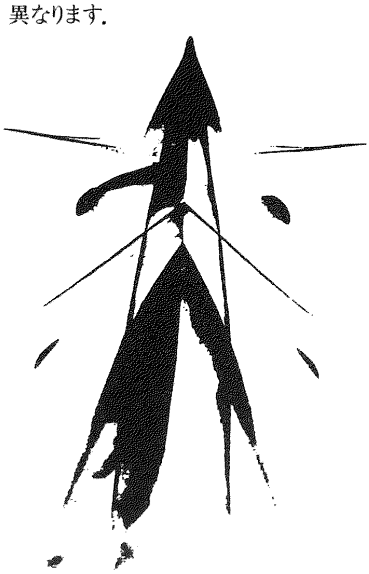


表３　MS-DOSシステムコール対応ライブラリ

INT番号	func.番号	Microsoft C	Turbo C
INT 21h	0Eh	_dos_setdrive	setdisk
		_chdrive	
	19h	_dos_getdrive	getdisk
		_getdrive	
	1Ah		setdta
	1Bh		getfatd
	1Ch		getfat
	25h	_dos_setvect	setvect
	27h		randbrd
	28h		randbwr
	29h		parsfnm
	2Ah	_dos_getdate	getdate
	2Bh	_dos_setdate	setdate
	2Ch	_dos_gettime	gettime
	2Dh	_dos_settime	settime
	2Eh		setverify
	2Fh		getdta
	31h	_dos_keep	keep
	33h		getcbrk
			setcbrk
	35h	_dos_getvect	getvect
	36h	_dos_getdiskfree	getdfree
	38h		country
	3Ch	_dos_creat	_creat
	3Dh	_dos_open	_open
	3Eh	_dos_close	_close
	3Fh	_dos_read	_read
	40h	_dos_write	_write
	43h	_dos_getfileattr	_chmod
		_dos_setfileattr	
	47h	_getdcwd	getcurdir
	48h	_dos_allocmem	allocmem
	49h	_dos_freemem	freemem
	4Ah	_dos_setblock	setblock
	4Eh	_dos_findfirst	findfirst
	4Fh	_dos_findnext	findnext
	54h		getverify
	57h	_dos_getftime	getftime
	57h	_dos_setftime	setftime
	59h	dosexterr	dosexterr
	5Ah		creattemp
	5Bh	_dos_creatnew	creatnew
	62h		getpsp
INT 23h			ctrlbrk
INT 24h		_harderr	harderr
		_hardresume	hardresume
		_hardretn	hardretn
INT 25h			absread
INT 26h			abswrite

I/Oポートおよびメモリアクセス関数
ポインタを使えばほとんど必要がない

8086系CPUでは，I/Oポートはメモリとは別の空間に割り当ててあり，専用の命令を使ってアクセスします．このため，Microsoft CやTurbo Cでは，I/Oポートへアクセスするための関数が用意されています．また，Turbo Cではメモリを直接操作するための関数も用意されています（表4）．

それぞれ，ワード（16ビット）単位でアクセスするか，バイト単位でアクセスするかによって2種類ずつ関数が用意されていますが，これも両処理系で関数名が異なってしまっています．

メモリの読み書きは，別に関数を使わなくてもC言語ではポインタを使って直接自由に行えます．したがって，peek，pokeはほとんど必要性がないものなのですが，Turbo Cがこれを用意しているのはBASICからCへ移ってきた初心者への配慮であると思われます．

表4　I/Oポートアクセス関数

機能	Microsoft C	Turbo C
I/O空間からの読み込み	inpw inp	inport inportb
I/O空間への書き込み	outpw outp	outport outportb
メモリの読み込み		peek peekb
メモリへの書き込み		poke pokeb

割り込みハンドラの記述
拡張キーワードを利用して実現

8086系CPUでは，割り込み（ハードまたはソフト）が発生すると，対応する割り込みベクタに記述されているアドレスに置かれた割り込みハンドラルーチンへ制御が移ります．割り込みハンドラは，必要な処理を終了したらIRET命令で復帰しなくてはなりません．したがって，Turbo Cの登場以前は，C言語で割り込みハンドラを記述することはできませんでした．通常，Cの関数はIRETではなくRET命令で復帰するからです．Turbo Cでは，interruptという拡張キーワードを指定した関数はIRET命令で復帰するため，割り込みハンドラの記述が可能です．また，Microsoft CでもV.5.1以降では_interruptという拡張キーワードで同様のことを可能にしています．

Cで記述した割り込みハンドラでは，ハンドラに飛んできたときの各レジスタの値を関数の引数として参照することができるようになっています．これは，各レジスタの値をスタック上に複製するコードが自動的に生成されるためです．そして，この複製はハンドラから復帰するときに元のレジスタに書き戻されますから，ハンドラ内でレジスタの値を変更することが可能になっています．また，割り込みハンドラ内では，DSの値は自動的にそのプログラムのDGROUPが設定されますので，プログラム内の変数などにもアクセスが可能です（Turbo Cのヒュージモデルでは，そのモジュールのデータセグメントの値がDSに設定されます）．

リスト6に，Microsoft CとTurbo Cによる割り込みハンドラの記述方法を示します．　見ての通り，まずキーワードが異なります．それにMicrosoft Cではfarキーワードも必須ですが，Turbo Cではfarをつけてはいけません．引数として扱える各レジスタの順序も異なります．Turbo Cのほうが各レジスタが順序よく並んでいるように見えますが，Microsoft Cの順序にはそれなりの理由

リスト6　割り込みハンドラ

Microsoft Cの割り込みハンドラ

```
void _interrupt far Handler(unsigned es, unsigned ds,
  unsigned di, unsigned si, unsigned bp, unsigned sp,
  unsigned bx, unsigned dx, unsigned cx, unsigned ax,
  unsigned ip, unsigned cs, unsigned flags)
{
  . . .
}
```

Turbo Cの割り込みハンドラ

```
void interrupt Handler(unsigned bp, unsigned di,
  unsigned si, unsigned ds, unsigned es, unsigned dx,
  unsigned cx, unsigned bx, unsigned ax, unsigned ip,
  unsigned cs, unsigned flags)
{
  . . .
}
```

があります．それは，80186以降（およびV30）のCPUで使用可能になった，全レジスタを一括して処理するPUSHA/POPAという命令が扱う順序に合わせてあるためです．したがって，Microsoft Cでは80186以降のコード生成を指定すると，PUSHA/POPAを使った効率のよいコードが生成されます．

Cでの簡単な割り込みハンドラの記述例をリスト7に示します．このプログラムは，INT 28hの割り込みハンドラとしてHandlerという関数を登録してから，COMMAND.COMを子プロセスとして起動します．INT 28hはMS-DOSの非公開割り込みで，キー入力待ちなどMS-DOSが暇なときに絶えず呼び出されているものです．これを利用して，このプログラムはPC-9801の画面左上に秒単位の時計を表示しています．INT 28hは，レジスタの値はとくに意味をもたないので，このハンドラでは引数として宣言していません．

両処理系で異なる点は，割り込みハンドラのキーワードが違う点と，時間を取得する関数（_dos_gettimeとgettime）の名前と使用する構造体が違う点，およびベクタの取得／設定を行う関数（_dos_getvectとgetvect，_dos_setvectとsetvect）の名前が違うという点だけですね．

一般にINT 28hのような便利な割り込みは，ほかのTSRプログラムなどがすでにフックして利用している場合があります．そのためこのプログラムでも，起動時に以前の割り込みハンドラのアドレスを取得しておき，Handlerの最後でそれを呼び出すように配慮しています．ハンドラからほかのハンドラを呼び出す場合は，Microsoft Cでは_chain_intrという関数が使えます．もちろんTurbo Cの例で行っているように直接呼び出すこともできます．この違いは，_chain_intrを使った場合はJMP命令で指定したハンドラへ移行するためHandlerには戻ってこないのに対して，直接呼び出した場合はHandlerに戻ってくるという点です．

リスト7　割り込みハンドラの記述例

(1)Microsoft Cの場合

```
#include <dos.h>
#include <process.h>
#include <stddef.h>

void (_interrupt far *lpOldHandler)();

void _interrupt far Handler()
{
    static short nBusyFlag = 0;
    static unsigned far *lpwVRAM = (unsigned far *)0xA0000000L;
    static char rgchBuf[9] = "00:00:00";
    static struct dostime_t Time;
    static int i;

    if (nBusyFlag == 0) {
        nBusyFlag = 1;
        _dos_gettime(&Time);
        rgchBuf[0] = Time.hour   / 10 + '0';
        rgchBuf[1] = Time.hour   % 10 + '0';
        rgchBuf[3] = Time.minute / 10 + '0';
        rgchBuf[4] = Time.minute % 10 + '0';
        rgchBuf[6] = Time.second / 10 + '0';
        rgchBuf[7] = Time.second % 10 + '0';
        for (i = 0; i < 8; ++i) {
            lpwVRAM[i] = rgchBuf[i];
            lpwVRAM[i+0x1000] = 0xA5;
        }
        nBusyFlag = 0;
    }
    _chain_intr(lpOldHandler);
}


main()
{
    lpOldHandler = _dos_getvect(0x28);
    _dos_setvect(0x28, Handler);
    spawnlp(P_WAIT, "COMMAND.COM", "COMMAND.COM", NULL);
    _dos_setvect(0x28, lpOldHandler);
}
```

(2)Turbo Cの場合

```
#include <dos.h>
#include <process.h>
#include <stddef.h>

void (interrupt *lpOldHandler)();

void interrupt Handler()
{
    static short nBusyFlag = 0;
    static unsigned far *lpwVRAM = (unsigned far *)0xA0000000L;
    static char rgchBuf[9] = "00:00:00";
    static struct time Time;
    static int i;

    if (nBusyFlag == 0) {
        nBusyFlag = 1;
        gettime(&Time);
        rgchBuf[0] = Time.ti_hour / 10 + '0';
        rgchBuf[1] = Time.ti_hour % 10 + '0';
        rgchBuf[3] = Time.ti_min  / 10 + '0';
        rgchBuf[4] = Time.ti_min  % 10 + '0';
        rgchBuf[6] = Time.ti_sec  / 10 + '0';
        rgchBuf[7] = Time.ti_sec  % 10 + '0';
        for (i = 0; i < 8; ++i) {
            lpwVRAM[i] = rgchBuf[i];
            lpwVRAM[i+0x1000] = 0xA5;
        }
        nBusyFlag = 0;
    }
    (*lpOldHandler)();
}


main()
{
    lpOldHandler = getvect(0x28);
    setvect(0x28, Handler);
    spawnlp(P_WAIT, "COMMAND.COM", "COMMAND.COM", NULL);
    setvect(0x28, lpOldHandler);
}
```

情報処理技術者試験
第2種合格！トレーニング
第2回　ハードウェアの基礎2

日本工学院専門学校　勝島貴之

今月は過去の試験で毎回出題されている「補助記憶装置の容量・処理能力計算」と，最近出題頻度が増加している「データ通信」を中心に解説します．

補助記憶装置の容量・処理能力

このテーマは過去の試験で毎回出題されています．磁気ディスク装置が中心ですが，磁気テープ装置，フレキシブルディスク装置などが取り上げられることもあります．

磁気テープ装置（MTE）

■容量計算

図1は磁気テープ(MT)上のデータの記憶状態を示したものです．入出力の効率を高めるために，複数のレコードをまとめてブロックという単位で処理を行います．1ブロックにまとめるレコード数を“ブロック化係数”，ブロックとブロックの間の無記録部分を“ブロック間隔（IBG）”と呼びます．

さて，容量計算で与えられる条件は**表1**のとおりです．この条件で，MTに記憶できるブロック数Bnは，レコード長・ブロック長の単位をテープ上に占める長さに変換し，テープの有効長内にどれだけのブロックが記憶可能かを計算することによって求めます．

公式1

レコード長 R_l [mm]
　$= r$ [バイト]／d [列／mm]
ブロック長 B_l [mm]
　$= n \cdot R_l$ [mm]＋ g [mm]
最大ブロック数 N_b [ブロック]
　$= L \cdot 10^3$ [mm]／B_l [mm]
最大レコード数 N_r [レコード]
　$= n \cdot N_b$

■処理能力計算

処理能力計算では，おもに**表2**のような条件が与えられます．

1ブロックの処理は，

入力処理(CPU処理時間＋MT処理時間)
↓
レコード処理（CPU処理時間）
↓
出力処理（CPU処理時間）

という流れになりますから，公式2が求まります．ただし，実際にはMTの停止とCPUの入出力処理・レコード処理などが並行に行われるため，ブロック処理時間の計算上は，MTの停止時間を無視する場合があります．したがって，問題の条件に十分注意して対処してください．

公式2

ブロック転送時間 T_{tr} [ms]
　$= n \cdot R_l$ [mm]／v [m/s]
ブロック読み取り時間
　T_{read} [ms]
　$= T_{start} + T_{stop} + T_{tr}$
ブロック処理時間 T_p [ms]
　$= T_{in} + T_{out} + n \cdot T_{rec} +$
　T_{read}

図1　MT上のデータ

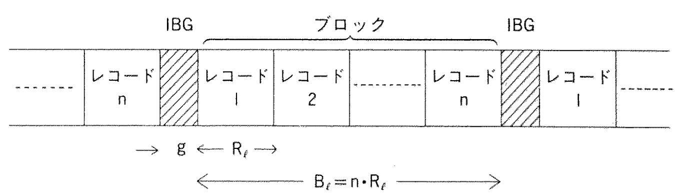


表1　MTE仕様（1）

項　目	仕　様
テープ有効長	L [m]
記録密度	d [列／mm]
レコード長	r [バイト]
ブロック間隔	g [mm]
ブロック化係数	n

表2　MTE仕様（2）

項　目	仕　様
テープ走行速度	v [m／s]
起動時間	Tstart [ms]
停止時間	Tstop [ms]
入力処理時間	Tin [ms／ブロック]
出力処理時間	Tout [ms／ブロック]
レコード処理時間	Trec [ms／レコード]

EXAMPLE 1

磁気テープ装置に関する次の記述中の［ ］に該当する値を求めよ.

記録密度64（列／mm），有効長720(m)のMTに，レコード長512(バイト)のデータを，1(レコード／ブロック)で記録すると［ ア ］(レコード)記憶でき，10（レコード／ブロック）で記録すると［ イ ］(レコード）記憶できる．ただし，ブロック間隔は16（mm）とする．

また，テープ走行速度が4(m/s)あるとき，10（レコード／ブロック）で格納した1ブロックを転送するのに，［ ウ ］(ms) 必要である．

解説

レコード長は$R_1 = 512/64 = 8$ (mm) です．ここでは，公式1を使います．

①1（レコード／ブロック）の場合

ブロック長$B_l = 1 \cdot 8 + 16 = 24$ (mm)

ブロック数$N_b = 720 \cdot 10^3 / 24 = 30,000$

レコード数$N_r = 1 \cdot 30,000 = 30,000$ …［ア］

②10（レコード／ブロック）の場合

$B_l = 10 \cdot 8 + 16 = 96$ (mm)

$N_b = 720 \cdot 10^3 / 96 = 7,500$

$N_r = 10 \cdot 7,500 = 75,000$ …［イ］

公式2のブロック転送時間を使います．

$T_{tr} = 10 \cdot 8$ (mm)$/4$ (m/s)

$= 20$ (ms) ……………［ウ］

EXERCISE 1

磁気テープ装置に関する次の記述の［ ］に入れるべき値を求めよ.

次に示すような仕様のMTEがある．

- ●記録密度　64［列/mm］
- ●ブロック間隔　15［mm］
- ●テープ走行速度　4［m/s］
- ●起動時間　3［ms］
- ●停止時間　3［ms］
- ●有効長　720［m］

(1) このMTEの最大記憶容量は，［ ア ］(Mバイト）である．

このMTEに，レコード長1024（バイト）のデータを1（レコード／ブロック)で記憶すると［ イ ］(レコード）格納でき，10（レコード／ブロック）で記憶すると［ ウ ］(レコード）格納できる．

(2) このMTEに，レコード長2048（バイト）のデータが10（レコード／ブロック）で10,000件格納されているとする．このとき，ブロック間隔を除いた1ブロックの長さは［ エ ］(mm)であるから，1ブロックを読みとるのに必要な時間は［ オ ］(ms)である．したがって，レコードの入力処理に要するCPU時間をブロック当り15(ms)，レコード処理に要するCPU時間をレコード当り1 (ms)（すなわち，ブロック当り［ カ ］(ms))，出力処理は無く，停止動作とCPUのレコード入力処理が並列動作するものとすれば，1ブロックの処理に要する時間は［ キ ］(ms)である．全データは［ ク ］(ブロック)であるから，全データの処理時間は，［ ケ ］(s) である．

解答

ア．46.08	イ．23,225
ウ．41,140	エ．320
オ．80	カ．10
キ．108	ク．1,000
ケ．108	

磁気ディスク装置

■容量計算

図2は磁気ディスク装置におけるデータの管理単位を示したものです．ディスクパックは数10枚のディスクから構成されます．個々のディスクには同心円状に数100の“トラック”があり，異なるディスク上で同一の円周位置にあるトラックの集合（円筒形状になる）を“シリンダ”として管理します．

MTEの場合と同様に，データはブロック化して記憶されます．注意が必要な点は，ブロックをトラックにまたがって記憶しないということです．

さて，**表3**は容量計算で与えられる条件です．計算には公式3を使います．

図2　磁気ディスク装置

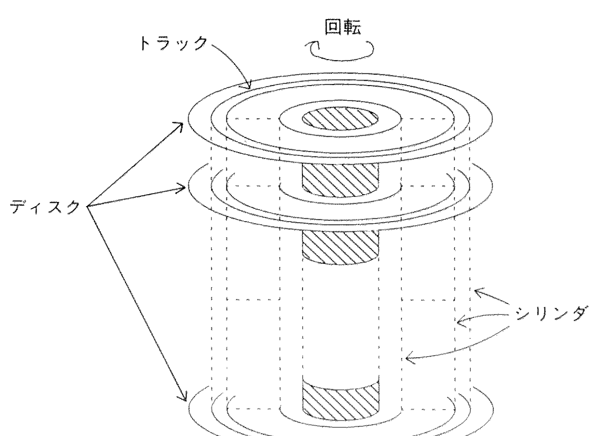


表3 磁気ディスク装置仕様(1)

項　目	仕　様
シリンダ数／ディスクパック	c［シリンダ］
トラック数／シリンダ	t［トラック］
トラック容量	m［バイト］
レコード長	r［バイト］
ブロック化係数	n
ブロック関数	g［バイト］

計算方法でMTEの場合と異なるのは，単位を長さで考えずにバイトのままで扱うことです．

公式3

ブロック容量 B［バイト］
　＝n・r［バイト］＋g［バイト］
トラック当りブロック数 N_t
　＝m［バイト］／B［バイト］
　　　　　　　　　…（＊）
シリンダ当りのブロック数 N_c
　＝t・N_t
ディスク当りのブロック数 N_d
　＝c・N_c
Sレコードを格納するのに必要なシリンダ数 N_s
　＝S／（n・N_c）　…（＊＊）
（＊）　小数点以下切捨て
（＊＊）小数点以下切上げ

■処理能力計算

ディスク上のデータへのアクセスは，
シーク（平均シーク時間）
　↓
回転待ち（平均回転待ち時間）
　↓
データ転送（データアクセス）
という順に行われます．

“シーク”でヘッドを該当シリンダ位置に移動し，トラック上の該当ブロックがヘッドの位置に到達するまで“回転待ち”した後，ブロックにアクセスします．

表4 磁気ディスク装置仕様(2)

項　目	仕　様
回転速度	v［rpm］
平均シーク時間	Tseek［ms］

さて，処理能力計算では主に**表4**のような条件が与えられます．計算には公式4を使います．

公式4

回転時間 T_r［ms］
　＝1000／（v［rpm］/60）
平均回転待ち時間 T_w［ms］
　＝T_r／2
データ転送速度
　V_{tr}［バイト／s］
　＝1000・m［バイト］／T_r［ms］
ブロックデータ転送時間
　T_{tr}［ms］
　＝1000・（n・r）／V_{tr}
ブロック平均アクセス時間
　T_{access}［ms］
　＝T_{seek}＋T_w＋T_{tr}

EXAMPLE 2

次のような磁気ディスク装置がある．
- ●シリンダ数／ディスクパック　800［シリンダ］
- ●トラック数／シリンダ　20［トラック］
- ●有効記憶容量／トラック　13,000［バイト］
- ●回転速度　3,000［rpm］
- ●ブロック間隔　235［バイト］

(1) 有効記憶容量を求めよ．
(2) 1024（バイト）のレコードを10万件格納したい．ブロック化を1（レコード／ブロック）で行う場合と6（レコード／ブロック）で行う場合の必要シリンダ数をそれぞれ求めよ．
(3) 平均回転待ち時間とデータ転送速度を求めよ．

解説

(1) 800・20・13,000＝208［Mバイト］
(2) 公式3を使います．
①1（レコード／ブロック）の場合
B＝1・1024＋235＝1,259［バイト］
N_t＝13,000／1,259＝10.3→10

N_c＝20・10＝200
N_s＝100,000／（1・200）
　＝500　→　500　…（答）
②6（レコード／ブロック）の場合
B＝6・1024＋235＝6,379［バイト］
N_t＝13,000／6,379＝2.04→2
N_c＝20・2＝40
N_s＝100,000／（6・40）
　＝416.7　→　417　…（答）
(3) 公式4を使います．
T_w＝T_r／2
　＝(1000／(3,000／60))／2
　＝20／2＝10［ms］
V_{tr}＝1000・13,000／20
　＝650,000［バイト／s］
　＝650［Kバイト／s］

EXERCISE 2

磁気ディスク装置に関する次の記述の［　］に入れるべき数値を求めよ．

次のような磁気ディスク装置がある．
- ●シリンダ数／ディスクパック　800［シリンダ］
- ●トラック数／シリンダ　19［トラック］
- ●有効記憶容量／トラック　24,000［バイト］
- ●回転速度　3,000［rpm］
- ●ブロック間隔　512［バイト］
- ●平均シーク時間　30［ms］

(1) この磁気ディスク装置の有効容量は［ア］(Mバイト）である．また，

この磁気ディスク装置は1回転するのに[イ](ms)かかるので，平均回転待ち時間は[ウ](ms)である．さらに，データ転送速度は[エ](Kバイト／s)である．

(2)この磁気ディスク装置にレコード長2048(バイト)のデータを1(レコード／ブロック)で記憶すると，1トラックに[オ](レコード)格納できるので，100,000(レコード)の格納には[カ](シリンダ)必要である．一方，10(レコード／ブロック)で記憶すると，1トラックに[キ](レコード)格納できるので，100,000(レコード)の格納には[ク](シリンダ)必要である．ただし，トラックにまたがってブロックを記憶しないこととする．

(3) 10(レコード／ブロック)で格納した場合の1ブロックのデータ転送時間は約[ケ](ms)だから，平均シーク時間と平均回転待ち時間をあわせた1ブロックの平均アクセス時間は[コ](ms)となる．したがって，制御プログラム，応用プログラムで要する時間を無視すれば，100,000(レコード)(すなわち，[サ](ブロック))の処理には[シ](s)要する．

解答

ア．364.8	イ．20	ウ．10
エ．1,200	オ．9	カ．585
キ．10	ク．527	ケ．17
コ．57	サ．10,000	シ．570

データ通信

従来データ通信に関する出題は1年に1回(春・秋のいずれか)でした．しかし，平成2年度には春と秋の2回とも出題され，この分野の重要性が増しつつあることがうかがえます．今までの出題内容は，データ通信システムの機器構成，データ通信サービス，伝送制御手順などに分類できます．

データ通信システムの基本構成

システムの基本的構成を**図3**に示します．使用する伝送回線の種類によって，システムを構成する装置が異なるので，次に解説する各装置の用途・機能をしっかりと理解しておきましょう．

■データ端末装置(DTE, Data Terminal Equipment)

端末側では，端末制御装置＋端末装置，センタ側では，通信制御装置＋計算機に相当する部分です．

■データ回線終端装置(DCE, Data Circuit terminating Equipment)

DCEの役割は伝送するデータの形式(信号形式)を伝送回線の性質・種類に適合させることです．したがって，アナログ伝送路の場合には変復調装置を，ディジタル伝送路の場合には宅内回線終端装置を用います．

●変復調装置(MODEM, MOdulator DEModulator)

ディジタル信号をアナログ信号で変調し，伝送路の使用可能周波数帯域にあわせて伝送したり，その逆の復調を行う装置です．

●宅内回線終端装置(DSU, Digital Service Unit)

DSUはDTEをディジタル回線に接続するためのインタフェースの役割を果たします．

■網制御装置(NCU, Network Control Unit)

公衆通信網(電話網，ディジタルデータ交換網，ISDN網など)を利用してデータ通信を行なう場合，相手DTEの呼び出し・接続ならびに切断に関して，公衆網内の交換設備の動作を制御するための装置です．

以上のようにデータ伝送回線の種類によって，DCEとして用いる装置の種類や，NCUの必要性が異なります．これらを**表5**にまとめておきましょう．

■通信制御装置(CCU, Communication Control Unit)

センタ側の処理装置に要求される主な機能は，①信号の同期，②伝送キャラクタの分解・組み立て，③誤り制御，④伝送制御，⑤メッセージ分解・組み立て，⑥コード変換，⑦アプリケーション処理などです．CCUは，このうち，①～⑥までの通信に係わる機能を果たす専用装置です．

図3　データ通信システムの基本構成

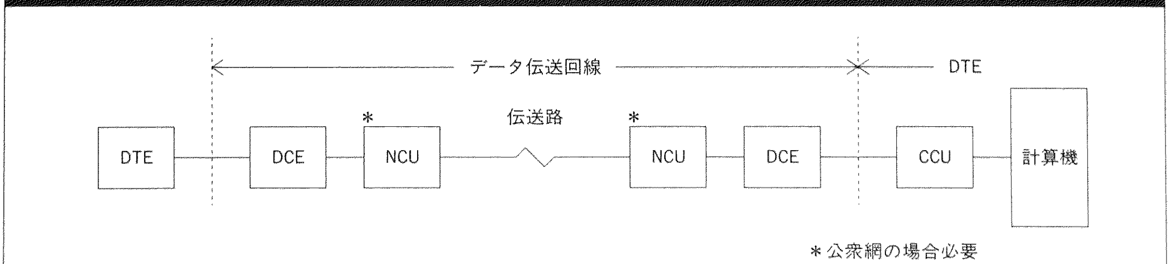


表5　DCEとNCUの使い分け

	アナログ	ディジタル
専用線	MODEM	DSU
公衆回線	MODEM+NCU	DSU+NCU

ディジタル公衆通信サービス

■ディジタルデータ交換網

（DDX, Digital Data eXchange）

回線交換（circuit switching）サービスとパケット交換（packet switching）サービスがあります．

回線交換は通信の開始から切断までの間，発着信端末間を結ぶエンド・トウ・エンドのディジタル伝送路を固定的に提供するもので，通信密度が高い場合に効果的です．

一方，パケット交換は蓄積交換に属します．データをパケット（packet）と呼ばれる単位に分割し，各々のパケットに宛先情報をヘッダとして付加して伝送します．

■サービス総合ディジタル網

（ISDN Integrated Services Digital Network）

従来個別網でサービスしていた音声・データ・画像などを，新しい伝送制御手順とディジタル通信技術により，1つのディジタル通信網に統合する方式です．

データ伝送制御手順

データを誤り無く，確実に送受信するために，データリンクレベルの通信規約（プロトコル，protocol）として，①データリンク確立・解放手順，②データ伝送手順，③誤り制御方式などが定められています．

■データリンク確立の方法

おもな手順として，センタが集中制御する“ポーリング／セレクティング方式”，分散制御で各端末が対等に制御権を持つ“コンテンション方式”があります．

■誤り検査符号

主な誤り検査符号として，パリティチェック方式とCRC方式があります．

●パリティチェック方式

データ内の“1”のビット数に応じて，検査ビットの内容を操作し，“1”の数をあらかじめ定めた偶数あるいは奇数のいずれかに保つ方式です．

●巡回冗長符号（CRC, Cyclic Redundancy Check）

被符号化対象の情報ビット $b_0b_1\cdots b_{k-1}$（kビット）を係数とする多項式

$F(X)=b_0+b_1X+\cdots+b_{k-1}X_{k-1}$

を，あらかじめ定められた生成多項式 $G(X)$（例えば，$X^{16}+X^{12}+X^5+1$）で割り，得られた余りの多項式の係数を冗長ビットとして $F(X)$ に追加して伝送する方式です．この符号は，連続したビットの誤り（いわゆる，バースト誤り）の検出能力に優れています．

■代表的なデータ伝送制御手順

データリンクレイヤの標準的なプロトコルには，文字（キャラクタ）コード伝送することを基本として比較的低速のデータ通信に利用される“基本型データ伝送制御手順（ベーシック手順）”，一方，コードインディペンデントな透過性（トランスペアレンシィ）や高速伝送を可能とする“ハイレベルデータリンク制御手順（HDLC手順，High level Data Link Control procedure）”などがあります．

EXERCISE 3

次の記述に最も関係の深い字句を，解答群から選択せよ．

a．電話回線を利用してデータ通信する場合に，信号をアナログ信号に変換したり，その逆を行なう必要がある．

b．ディジタル通信サービスで必要な装置で，伝送速度の変換，制御ビットの挿入，信号形式の変換などを行なう．

c．通信回線とコンピュータを接続する装置で，主に文字分解・組み立て，誤り制御機能などを有する．

d．信号はフレームと呼ばれる単位で扱われ，伝送する情報内容がコードに依存せず，高速伝送が可能な伝送制御手順である．

e．データをある大きさに区切って，宛先情報，順序番号などを付与し伝送する．ネットワーク内に蓄積交換機能を持つため，異なる通信速度の端末間の通信も可能である．

f．通信を行うには様々な約束，規約を定める必要がある．

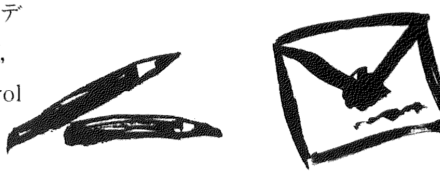
g．誤り検出のための符号である．伝送データを多項式とみなし，あらかじめ定めた生成多項式で割った余りを，冗長ビットとして付加する方式である．

<解答群>

ア．パケット交換　イ．ISDN
ウ．CCU　エ．DSU　オ．LAN
カ．CRC　キ．MODEM　ク．NCU
ケ．HDLC　コ．プロトコル

解答

a－キ，b－エ，c－ウ，d－ケ，
e－ア，f－コ，g－カ



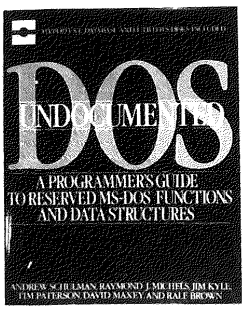
Book Review

■中村正三郎

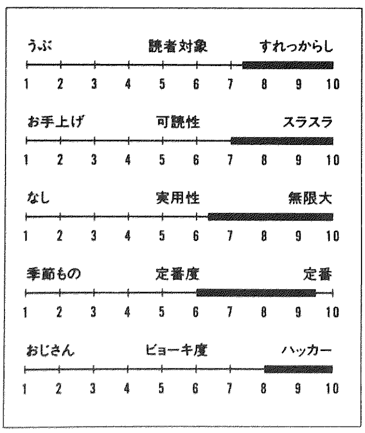
DOSハッカーたちがまとめ上げたMS-DOSの非公開ファンクション

■Undocumented DOS

Andrew Schulman, Raymond J. Michels, Jim Kyle, Tim Paterson, David Maxey, Ralf Brown 著
Addison Wesley



■評価チャート



本書は，タイトル通りドキュメント化されていない，つまりMS-DOSの非公開ファンクションを解析し，解説した本である．

これらの情報のいくつかは，我が国のパソコン通信などでも散発的に解析結果が流れていることはある．しかし，これほどまとめ上げて本にしたのは，本書がはじめてではないか．

一通り眺めただけだが，よくもまあ調べ上げたものだと感心してしまった．著者たちはいずれもDOSハッカーであり，米国の雑誌でも著名な人や，Tim Pattersonのように最初のMS-DOSを作った人も含まれており，情報の確度は高いといえるだろう．

基本的にはPC-DOSの世界についての記述であるから，本書の内容がそのまま日本のDOSにあてはまるかどうかはわからない．読者で検証していただきたいが，多くは大丈夫だろうというのが現在の私の印象である．日本のDOSにあてはまらない記述についても，DOS/Vが騒がれている昨今ではあるし，有益なものだと思う．

本書の構成は，次のようになっている．第1章は，非公開ファンクションを使う意義．第2章は，公開ファンクションと非公開ファンクションの比較．第3章からは各論になる．第3章は，メモリ，プロセス，デバイスといったDOSのリソース管理．第4章はファイルシステムとネットワークのリダイレクタ．第5章は常駐型ソフトについて．ここではマルチタスク型TSRの話もある．第6章は，COMMAND.COMについて．第7章はMS-DOSのデバッガインターフェースについて．第8章は，これ以前の章でDOS内部を探るために使われているプログラムINTRSPYに関する説明である．付録Aが，非公開ファンクション一覧になっており，本書の3分の1ほどの分量を占める．付録Bは注釈付きの参考文献一覧である．PC MagazineやDr. Dobb's Journalなどの雑誌記事も紹介してある．

各章では，非公開ファンクションの説明とともにソースプログラムを示しながら，その使い方を述べていく．さらには，非公開ファンクションを用いたユーティリティ，たとえばデバイスをロードするDEVLODやメモリチェーンを表示するMEMなどを作っている．これらのユーティリティはソースとともに実行形式が添付のディスクに入っている．

ここで本書には，ディスクが2枚添付されていることに注目したい．これには，先に述べたユーティリティ類のほかに，ハイパーテキスト風のオンラインドキュメントが入っている．これがとてもよい．この閲覧プログラムINTRLISTには，本書に述べられている非公開ファンクションはもとより，Novellのファンクション，BIOSのデータエリア，CGAやVGAの情報，INT 21H以外の各種割り込み情報な

どが満載してある．これだけでもDOSプログラマには重宝する．

以上，本書を概観した．熟練したDOSプログラマにはすでに知っている情報が多いかもしれないが，それでもこれだけまとまった形で入手できることは，大いに購買意欲をそそる．とくに，INTRLISTは多くのDOSプログラマにとって有用だろう．

動きはじめたMacintoshに期待

古田島義和（神田・書泉グランデ）

今年2月に幕張メッセで開催されたMACWORLDExpo/Tokyo'91以来，Macintosh関連図書の動きがよくなってきました．昨年はMacの世界でも10万円台の機種が発売され，今後Mac人口がどれくらい増えるか楽しみです．

ここでMac図書を数点紹介しましょう．3月に「ResEdit Complete」(Addison Wesley／以下AW)が出版されました．以前「ResEdit Reference」が出版されて大好評でしたが，今回のは，プログラマやエンドユーザーにも必見の1冊で，発売以来前回をはるかに上回る人気ぶりです．ロングセラーに「Inside Macintosh」シリーズ全5巻（AW）があり，約30万部以上売り上げています．これは日本語版も出版されています．

そしてシリーズ第6巻目が91年6月に発売予定になっています．今回は，System 7を取り入れての解説書です．これと同時期に「System 7 Revealed」「Inside Macintosh X-Ref」「Inside the Macintosh Communications Toolbox」(3冊ともAW／いずれも'91年6月頃出版予定）が続々と出版されます．

今年，Wolfram氏がMathematica Ver. 2. 0を発表，本格的に動き出しました．Mathematicaは，各主要コンピュータ(Mac，IBM，NeXT，HP，SUN，Sony，DEC）のマシンで動くソフトウェアとして期待されています．Mathematica関するおもな図書は，次の通りです．開発者であるWolfram氏の著書「Mathematica 2/E」(AW）をはじめとして，「Programming in Mathematica 2/E」(Maeder)，「Mathematica for the Science」(NeXT ver.／Crandall)，「Exploring Mathematics with Mathematica」(with CD-ROM／Gray)，「The Mathematica Graphics Guide Book」(Smith)，「Calculus & Mathematica Part 1」(Brown)，季刊雑誌「Mathematica Journal 90 Summer, 90 Fall」が好評発売中で近日中にSpringが入荷予定です．日本では，Macintosh Smart Guide3「Mathematica 数式処理入門」(技術評論社)が出版，これはMac用に書き下したものです．今後Wolfram氏の「Mathematica Ver. 2. 0」の訳本が出版される予定です．

今年出版各社が力をいれているのが，Network関連図書です．Networkの代表的なもののひとつである「Netware」は，Novell社が開発したもので，米国で非常に多くの人が使っているNetです．日本でも「Netware 386J」という日本語版が出荷されました．しかし，これは3月にバグが発見され，少々不安なところですが，解説書は順調な売れ行きを示しています．まだ386Jに対応する日本語解説書がないので洋書を紹介します．「Netware 386 Programmer's Guide」(AW)，「Netware 386 User's Guide」「Netware Programmer's Guide」「Netware Supervisor's Guide」「Netware Decorder」(4冊ともPrentice Hall)，「Netware Trouble Shooting」(AW／6月出版予定)，以上が386に対応する図書です．ほかにも，286とかNetwareに関する図書が今年中に約20点以上予定されています．

最後にMicrosoft Press社から，注目の新刊2点を紹介します．「Object-Oriented Programming Power for Think Pascal Programmer's」(Sphar著）と「Macintosh C Programming Primer」です．AW以外からこの手の図書がでるのは驚きです．

■和書&洋書ベスト10　　（書泉グランデ調べ和書：1991年2月1日～2月～28日）

	和書	出版社
1	VZ倶楽部	ビレッジセンター
2	rootから／へのメッセージ	アスキー
3	コンサルタントの秘密	共立出版
4	プログラミング言語 C 第2版	共立出版
5	MS-Windowsとは何か	講談社
6	実習C言語 改訂新版	アスキー
7	PC-9801シリーズマシン語ゲームプログラミング	アスキー
8	秘伝 C言語問答 ポインタ編	ソフトバンク
9	Turbo C++オブジェクト指向プログラミング入門	技報堂
10	VZを256倍使うための本	アスキー

	洋書	出版社
1	Undocumented DOS (with Disk)	Addison Wesley
2	Compilers (paper)	Addison Wesley
3	ResEdit Complete (with Disk)	Addison Wesley
4	Simple Book:An Introduction to Management of TCP/IP	Prentice Hall
5	System Performance Tuning	O'Reilly
6	Mathematica for Science	Addison Wesley
7	UNIX Network Programming	Prentice Hall
8	OSF/Motif Programmer's Guide Release 1.1	Prentice Hall
9	Object-Oriented Analysis 2/E	Prentice Hall
10	Macintosh Bible Super Combination 3/E (Disk/Books package)	Goldstein & Blair

ワンラインNEWS
タッチパネル・システムズ社はマウスのかわりにタッチパネルを使えるようにする「モニターマウス」を発売した．IBM PC，PS/2対応で価格は8万円．

◆ ◆ 佐々木哲のニューヨーク ◆ ◆

N.Y. システム通信

ソ連の状況悪化で蜜月の終わり

以前このコラムで書いたように，一時ソ連をはじめ東欧ブロックのコンピュータをとりまく環境はとてもよいものになりかけていました．もちろんソ連のグラスノスチに従った東西緊張の緩和の結果です．

ところが，このところのバルト情勢などの社会不安，保守反動，それに伴う経済不安の波にのまれ後退しています．コピー機の自由な所有も認められないソ連．そこでは画期的なニュースサービスだったInterfaxも規制を受けているようです．コンピュータにとって最大の壁だったCOCOM規制が緩和されるという話もとりやめになりました．各種機器の値段も20%くらい上がったという話もあります．国内のコンピュータメーカーはヤミ食糧で部品の購入にあたる始末．蜜月の終わりですね．やはり現実が待っています．自由がないところにはコンピュータ文化はありえません．それでも西欧各コンピュータ関連企業のソ連進出は着実に進んでいるようですが．

まだまだおもしろいパソコンFAX

先月パソコン用のFAXモデムについて概略を書きました．いま問題となっていることは，FAXモデム一般のプログラムインターフェースの欠如です．一般のパソコン通信のモデムならば，どれを買ってきてもHayes互換のコマンドをサポートしています．自分の好みのソフトで使うことができます．ところがFAXモデムでは，それぞれ各ボード専用の添付ソフトでしか使うことができないのです．これはちょっと融通がきかないめんどうなことですね．使い勝手の悪いソフト付きのFAXモデムを買った日には目も当てられません．

日本のJISに相当するアメリカのEIAでは，TR29としてFAXモデムのAPIの共通化をはかっています．FAXモデムが一般のHayes互換モデムと同様のATコマンドで，FAXの送受信ができるようになるのです．これは画期的なことです．自分の使いやすいFAX通信ソフトを選んで使えるようになる．ProcommやCrossTalkなどの一般の通信ソフトでFAXを送れることになるのかもしれません．一般のアプリケーションも，このAPIさえサポートしておけば，どのFAXモデムからでもFAXが送れるようになります．ユーザーにとってもアプリケーションの開発側にとってもとても便利になります．

このTR29には，いくつかクラスがありclass 1は決まりましたが，class 2とたぶん3はまだ完全には固まっていないようです．決まったあとは，CCITTにかけられ世界的な規格となるでしょう．

パソコン用のものとしては，class 2と呼ばれるものが広まると予想されます．どのFAXモデムを買ってきても，class 2のAPIをサポートするソフトならば使えるようになります．

これをサポートした実際の製品はまだごく限られています．モデムのチップの代表的なメーカーはすでに開発をはじめています．たとえば，Rockwellは2チップで2400bpsのモデムと9600bpsのclass 1，2インターフェースをもったチップセットを発表しています．今年はいろいろ発表されることでしょう．

究極の通信ソフトMTE

いまでは，MNP付きのモデムは当たり前になってきました．やはり文字化けを心配することなしに通信できるのはいいものです．MNPをソフトだけでやってしまおうという通信ソフトがいくつかあります．その代表的なものがMagicSoftのMTE*1．すでに市場に出て3年近くなるものです．単体でのほかに，多くのMNPなしモデムに添付しても販売されています．以前はINTELのMNPなしモデムにも付いていました．

ソフト自体はProcommを意識したデザインで，ソフトウェアだけでMNP5を実現しています．つまり種も仕掛もない非MNPのHayes互換モデムでMNP5のエラーフリー，圧縮付きの通信ができるのです．

これは別売りオプションが豊富なソフトです．Emulation Add-Onパッケージを買うと，ZMODEMどころかIBMのメインフレーム用IND$FILEまでのファイル転送プロトコルや，多数の端末エミュレーションが含まれています．また，Cellular Module Add-Onもあります．これは非常に過酷な携帯電話の環境でも，接続を失

ワンラインNEWS
日本クレイはスーパーコンピュータの市場開拓のため営業体制を大幅に強化する．技術支援に特化した社長直属の新組織「営業技術グループ」を発足させた．

うことなしにエラーフリーの通信を可能にしています.

さらに，最近ExpressFaxというモジュールを発売しました.これは先月のWinFAXと同様に，Sierraチップを使った送信専用のFAXモデム，あるいはClass 2規格をサポートしたFAXモデムでFAXが送信可能になるものです.INTELのCASもサポートしています.

通信ソフトからファイル転送と同じ感覚でFAXが送れるというのも素直な発想でしょう.

なお，このMNPのエミュレーション機能だけをサポートした通信モジュールMX5も別売されています.IBM互換機BIOSのINT 14hインターフェースを拡張したFossilドライバとなっています.CrossTalkなど，これをサポートした通信ソフトで使えます.

ここもはっきりした規格が望まれるところなのですが，通信ソフト側で実際の通信モジュールの切り口をはっきりさせておき，切り替え可能にしておけば，このMNPモジュールやLANなどへの対応がとても楽です.

＊1：MTE $79.00
Magicsoft Inc., P.O.Box 396 Lombard, IL 60148
phone (708) 953-2374, fax (708) 953-2375

身近なマルチメディア

昨年あたりからマルチメディアという言葉がはやっています.いったい何なのか，まだ誰もわからない状況です.いままでのディスプレイでの文字の表示だけではなく，音声，動画などを付加したものととらえればいいのかな？　とても広い範囲を含むものです.

パソコンがしゃべるだけでもインパクトはあります.たとえば，WordPerfectのようなワープロや，Lotus 1-2-3に音声が録音できたらどうですか？　単なる画面上の文章や表をよりわかりやすくでき，くらべものにならないほど説得力のあるものにできますね.

Voice-It[*2]というソフトがそれを可能にしました.WordPerfect 5.1とLotus 1-2-3のadd onとして動作します.必要なハードウェアは簡易なLAN，ArtisoftのLANtastic用のVoice Adapterです.WordPerfectの文章や，1-2-3のセルデータといっしょに音声を記録できるのです.

メッセージを録音するにはWordPerfectかLotus 1-2-3で，そのデータのあるパラグラフかセルにカーソルをもっていきます.そしてVoice Adapterのマイクからコメントや説明などを録音します.再生するときには，play backを指定するとメッセージを再生しながらそのパラグラフやセルが順次表示されます.

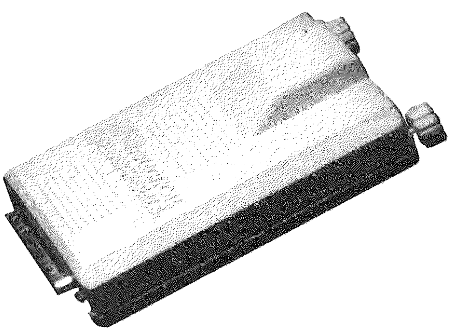
文章で書いただけでは，なかなか相手に伝わりにくいこともより明快になります.効果のあるプレゼンテーションが可能になりますね.データファイルは，LAN上でほかのユーザーに送ってもいいし，ファイルを送ってもいいわけです.あなたならどう使いますか？

＊2：Voice-It $199 with Voice Adapter, $99 software only
Voice-It,Inc., 1001 Snowden Farm Road, Collierville, TN 38017
phone (901) 853-4456

今月の何だこりゃ？

先月ご紹介した格安のV.32モデムを出しているPractical Peripherals Inc.，もうひとつとんでもない携帯用モデムを昨秋から販売しています.その名もPocket modem[*3].値段はたったの159ドル.実際には100ドル程度で買えます.大きさは，タバコの箱よりひと回り小さいというか，普通の25pinシリアルケーブルコネクタをひと回り大きくした程度です.2400bpsのHayes互換モデム.MNPなしです.スピーカーもありません.ランプさえもいっさいついていません.それどころか電源がついていない.そう，電源不要のモデムなのです.消費電力がとても少ないようにデザインされ，RS-232Cインターフェースと電話線から電力をとって動作します.夜中に電池が切れて大あわてする心配もありません.ラップトップに最適ですね.上で紹介したMTEといっしょに使えば，MNP5のエラーフリーで通信できます.

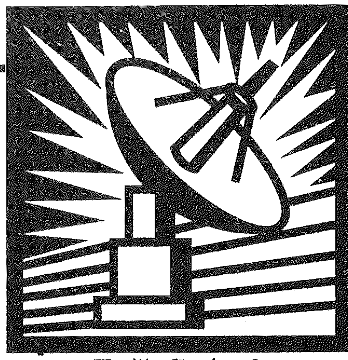
＊3：Practical Pocket Modem PM2400PPM $159.00
Practical Peripherals,Inc., 31245 La Baya Drive, Westlake Village, CA 91362
phone (818) 706-0333, fax (818) 706-2474



写真左端が25ピンのコネクタだから，だいたいのサイズがわかる.

SYSTEM N.Y. NEWS

Illustration : NOBUKI FUKUHARA



西川和久の

希望的観測

米国ビデオ規格用語乱用に問題あり!!
OADGや日本語Windows3.0に見られる画面解像度戦争

1 640×480≠VGA 1024×768≠XGA

最近OADG（PCオープン・アーキテクチャ推進協議会）や日本語Windows3.0が注目されるに従って米国PCにまつわる話，とくにビデオ規格の話がいろいろな雑誌や新聞などでとりあげられている．個人的には，非常に喜ばしいことだと感じているが，その記述や表現に問題があるように思われる．まず，画面解像度を語るとき，おかしな表現をしている場合が多い．読者に「画面の解像度が640×480はVGA，1024×768がXGA」と思わせてしまう文章が随所に現れるのだ．たとえば，「FM-TOWNSはVGAを採用しているため……」とか「PS/55に搭載しているXGAは……」など，あきらかにライターが勘違いしているとしか思えない表現が見受けられる．もっとひどいものでは「PC-9801はEGAを採用しているため……」などという，NECが見れば激怒しそうな文章まであった．

そもそもCGA*1，EGA*2，VGA*3，XGA*4と呼ばれている各Video規格は，米国のIBM PCが歩んできた10年間の軌跡だといっても過言ではない．これらは画面の解像度だけではなく，『いろいろなI/OレジスタやBIOSのエントリーまで，すべて含めたものの総称』なのである．このような背景を明らかに無視，または知らずに，画面解像度だけを大きくクローズアップしてものごとを語るのは，読者を混乱させるだけである．今年になってから，ドタバタと米国PCに関係のある発表が続いたために混乱するのもわからないではないが，ライターと名がつく限り，もう少し責任をもって読者に誤解のない文章を書いていただきたいものである．

＊1 CGA:Color Graphics Adapter
＊2 EGA:Enhanced Graphics Adapter
＊3 VGA:Video Graphics Array
＊4 XGA:Extended Graphics Array

2 640×480≠マルチメディア対応

次は，『マルチメディアと解像度』のことである．マルチメディアの話題は，いまのところ爆発するまでには至っていないが，これからのシステム環境として最近よくとりあげられている．ところが，これについてもまた画面解像度から判断して「640×480なら問題ない」と勘違いしているのだ．困ったものである．

たしかに，画素の比率が１：１になるため，一般的なビデオ画像を扱うには最適な解像度ではあるが，やはりVGAなしでは考えにくい．理由は簡単で，現在開発されているほとんどのマルチメディアシステムは，VGAと何らかの形で同期をとりながら構築されているためである．いいかえれば，VGAという規格とマルチメディアは切っても切れない縁なのである．ところが，先ほどと同様に画面解像度だけが一人歩きして，肝心のところが抜けてしまっている．どこかの広告でありました

ワンラインNEWS
米ラディウス社はMacintoshII用のPアクセラレータボードを４月に日本市場で発売する．CPUに68040を搭載したボードで処理が数倍になる．

ね「クリープを入れないコーヒーなんて……」っていうのが…… :-)

3 「Windows 400」なんですかそれ!?

今回のタイトルからは話が少しそれてしまうが，「画面解像度戦争」に含まれるのでとりあげようと思う．くれぐれも誤解のないよう，はじめにお断りしておくが，「Windows Version 400.0 (???)」の話ではない．%-)

少し前にあった，Windowsコンソーシアムの会場でNECの方がスピーチされ，そのときに出てきた言葉がこれである．まだ企画段階らしいのだが，なんと『640×400の解像度をもつ日本語Windows3.0のためのコンソーシアム』の意!! これには正直いって驚いてしまった．同じ話の中で「米国のWindowsアプリケーションを640×400の日本語Windows3.0で使った場合，上下が切れてしまう問題点はあるが……」と明言した上でのこの発言．強気というか，ユーザーを馬鹿にしているというか，とにかく正直いってまいってしまった……．

おもしろいことに (!?)，この「Windows 400」なるものに加入すると予想された東芝，松下などのOADGへの登録の意向が，3月11日に正式に発表された．このことによって，日本IBMやAXを含めた『640×480』VS『640×400』の構図ができ上がってしまったのだ．今後，米国PC互換機メーカーもこのOADGに参入してくることは明白であり，これまでには考えられないほど壮絶な戦いになるであろうと予想される．湾岸戦争の構図にも似ているこの画面解像度戦争，どちらが独裁者率いるイラク軍でどちらが他国籍軍かはあえていわないが，ともあれ，どう考えてもたった2か月程度で決着するとは思えない雰囲気である %-)．

4 パソコン文化の違いが混乱を招いている

この画面解像度戦争，そもそも国産パソコン文化においてディスプレイカードが固定されてしまい，ユーザーやシステムのリクエストに対してフレキシブルに対応できない設計になっているのが大きな問題点なのである．そこにOADGや日本語Windows3.0，XGAなどの登場による米国PC文化の短期間大量輸入．これでは混乱するのも当然である．

なぜ，米国では，CGA，EGA，VGA，XGAと10年の歳月を経て320×200から1024×768になり，ディスプレイ回路はカード形式にする必要性があったのか?? また，なぜ国内では，ほぼ同じ歳月がたっているにも関わらず，主力は640×4?0のままで，かつマザーボードに固定なのか?? よく「10年ひと区切り」といわれるが，ちょうどいま，見直しをはかる時期にきているのではないだろうか．「ソフトがなければただの箱」などとコンピュータを指す言葉としてよく使われるが，やはりハードも大切である．OSばかりを日本語化するのではなく，米国のハードに関する状況を見習い，いろいろな環境が進化する中で，ユーザーの負担をなるべく少なくするために，メーカー側の努力が必要になっているのではないだろうか!?



Computer Laws

コンピュータの法律

バックアップ用のコピーに関する問題点

◆

熊谷信太郎(弁護士)

今回は，前回に引き続いて，プログラムのバックアップ用のコピーに関する具体的な問題を検討したいと思います．

■著作権者の承諾がないとだめ

市販のプログラムを購入した後に，「自分で使用する限度において」，プログラムのバックアップ用のコピーをとることができることについては，前回に説明したとおりです*1．

ところで，著作権法は，市販のプログラムを購入した者がプログラムのバックアップ用のコピーをとっている場合に，本物のプログラムについて，「滅失以外の事由により」所有権を失った場合には，その者は当該著作権者の別段の意志表示がないかぎり，バックアップ用のコピーを保存してはならないと定めています*2．

■ところで「滅失」とは

では，「滅失以外の事由により」，本物のプログラムの所有権を失うというのは，どのような場合でしょうか．

本物のプログラムが壊れて使えなくなった場合に備えて，バックアップ用のコピーをとるのですから，本物のプログラムが壊れることが，ここにいう「滅失」にあたることは間違いありません．したがって，本物のプログラムが壊れた場合に，バックアップ用のコピーを保存して，それを使うことができることは，いうまでもありません．

■他人へ譲るとき

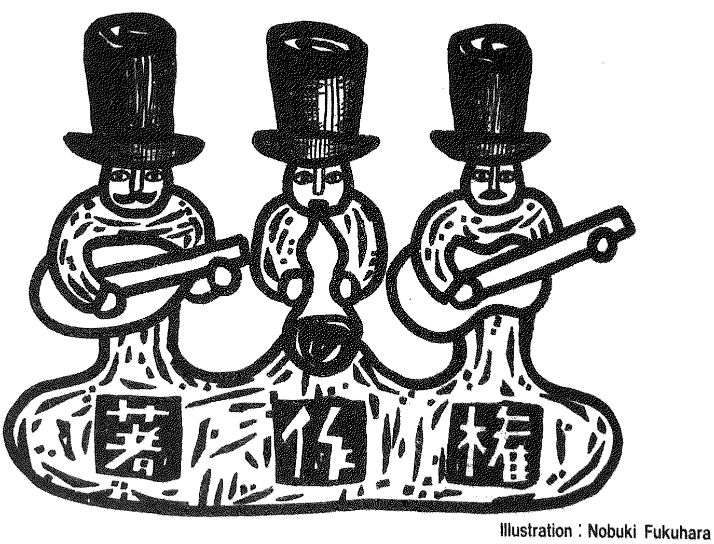
これに対し，本物のプログラムを他人に譲渡する際に，バックアップ用のコピーは譲渡せずに，自分でとっておいて，その後に自分でバックアップ用のコピーの方を使うという行為はどうでしょうか．

他人への譲渡により，本物のプログラムの所有権を失うことは，ここにいう「滅失」にはあたらないと解されています．というのは，バックアップ用コピーというのは，あくまで，本物のプログラムが壊れる，すなわち「滅失」したときに，バックアップするためのものであり，本物のプログラムの所有権が失われるすべての場合にそなえてバックアップが認められているわけではないからです．

したがって，このような場合にバックアップ用のコピーを保存することは著作権法第47条の2第2項に違反することになり，バックアップ用のコピーは廃棄しなければなりません（もっとも，この規定に違反して，バックアップ用のコピーを保存しても，罰則はありません）．

■盗まれたとき

また，他人に本物のプログラムを盗まれたり，横領された場合に，手元にバックアップ用のコピーが残っていれば，ほっと一安心というところですが，これも著作権法の建前からいうと，本



Illustration：Nobuki Fukuhara

物のプログラムが「滅失」したわけではなく，単に所有者の意志に反して所有権が移転したにすぎないので，手元にあるバックアップ用のコピーをそのまま保存して使うことは許されず，廃棄しなければなりません．

しかし，このような結論は，他人に本物のプログラムを盗まれたり，横領された人にとっては，あまりに酷な結果となるので，著作権法第47条の２第２項は，このような場合，窃盗や横領の被害者は，著作権者の別段の意思表示があれば，すなわち，著作権者の承諾を得れば，そのままバックアップ用のコピーを保存してもよい（したがって，当然，使用してもよい）と定めています．

＊1　著作権法第47条の２第１項
＊2　著作権法第47条の２第２項

アキバ短信

ダニエル
鉄板に並ぶ54個の玉子を見よ

電気のデパート秋葉原には電気製品とは無縁のデパートもある．JR秋葉原駅に隣接した（というよりも正確には総武線ホームの真下に位置する）駅ビル「アキハバラデパート」である．電気の街秋葉原にあって，洋服（紳士服から女性の下着まで），雑貨，食料品，書籍（一般の雑誌や文庫本），薬まで売っているので，時間のないとき，あるいは時間のあり余るときには何かと重宝する．

そんなアキハバラデパート１階には多数の飲食店が入っている．カレースタンドのインドール，丼専門店の伊呂波，寿司岩，讃岐うどんの古市庵，ハンバーガースタンドのmorinagaなど手軽に食べられる店が多い．数ある中で今回は，たこ焼き，お好み焼き，焼きそばの店ダニエルを紹介しよう．

ダニエルのメニューはたこ焼き，お好み焼き，焼きそばとそのバリエーションである．お好み焼きそば（お好み焼き＋焼きそば），たこ焼きそば（たこ焼き＋焼きそば），肉入り焼きそば（これは具を焼いて焼きそばに混ぜる．焼きそばのデフォルトの具は野菜）などがその代表だ．でも，ダニエルへ行くのがはじめてならお勧めはお好み焼きだ．その主たる具はキャベツと玉子．ちょうどお好み焼きの間にキャベツと目玉焼きがサンドイッチになっていると思ってもらえばいい．これに甘めのたれを好みの量だけハケで塗ってワリ箸で食べる．



営業時間　10:30〜19:00　無休
アキハバラデパート１階

お好み焼きを食するときの最大のポイントは「立ち食い」．奥の方にカウンター席がいくつかあるのだが，辺りで立ったまま食べている人も多い．さりげなく鉄板の前に立ってお好み焼きのできるまでのプロセスを鑑賞するのだ．大まかにいってそれには５つの段階がある．

1. タネを鉄板いっぱいに流す
2. その上にキャベツをのせる
3. 縦に６段，横に９列計54個の玉子を割入れる
4. 縦に６段，横に９列に切り分けて裏返す
5. 裏面を焼く

これで鉄板いっぱいに54枚の四角いお好み焼きができあがる．２と３の間ではきれいに玉子を割入れるためにキャベツを均等にならす，４の裏返しのプロセスでは広い鉄板の中心と周辺の火力の違いを考慮して裏返す順序に変化をつけるといった多少の技術の入りこむ余地も見られるが，このお好み焼きの基本的なポリシーは鉄板の面積を最大限に活かした「大量生産」にあると見た．54個の玉子を割入れ，54枚の四角いお好み焼きを裏返す，その繰り返しは，あたかも教育テレビ「はたらくおじさん」に出て来る町工場を思わせる．

だからといって，無機的なお店を想像するなかれ．「あれ？このたこ焼きたこが入ってないよー」とボソっとこぼすと店の人がすかさず１個オマケしてくれる．そんなやり取りがよく似合う店だ．くだんのお好み焼きはおみやげにもできる．

プログラマの仕事場 | 2

Programmer's Working Space

株式会社ミワシステムズコンサルティング

UNIXからMS-DOSまでをフォローするC++のパイオニア

いまや「オブジェクト指向」という言葉はトレンドとなりつつあるが，実際にオブジェクト指向によるプログラムの設計，開発を行っているプログラマはごくわずかだろう．ミワシステムズコンサルティングは，アプリケーション開発言語として早くからC++に注目し，C++の日本語開発環境ごと自社で作り出したという，日本でのC++プログラミングのパイオニア的存在である．同社企画開発部を訪ねて，取締役部長足立高徳氏にお話をうかがった．

前身は電気工事会社の開発部門

「MIWA C++」というC++トランスレータを開発したことで知られるミワシステムズコンサルティングだが，意外なことにもとは電気工事会社の開発部門だという．

「親会社も建設業界のなかでは小さな会社ですし，その中の開発部門が母体ですからシステムの開発をする人間は最初は2人くらいでした．そのころはもちろんC++もありませんでしたし，販売するだけの力もありませんでした」と足立氏．

「最初の積算システムの開発はBASICを使いました．当時はDOSの上で動くCといってもLattice Cしかなかったですしね．使っていたマシンも98の初代とかEとかでした．ハードディスクもなくて，フロッピーは当然8インチですし，いまにして思えば劣悪な環境でした．Cはまだ重かったので，BASICでということになりましたが，言語としてのポテンシャルも低いですし，わりと早いうちにCにかえたんです．

それを使っていたら仲間うちの工事屋さんから『何を使っているの?』と．当時はまだワープロの文字もめずらしい時代でしたからね．これはパソコンだよといって説明すると，これならひとつ欲しいという具合いだったんです．最初はソフトはただでした(笑)．そのうちサポートをして欲しいという話がきて，ただでは気が引けるからお金を取ってほしいと，外からいってきたんです．それなら売ろうかという話になったところ，急にバァーッと売れた．そうすると今度は経理の方から工事代金でもないのに困るといわれまして，そこで会社を作ったというわけです．現在でも製品売上のおよそ65パーセントは業務用ソフトです」．

もとは自分たちの開発用だったC++

同社では87年から開発用言語としてC++を採用している．

「はじめはCで開発をしていましたが，4年ほど前から少しずつC++にシフトしていきました．C++を使うことにした直接の動機は『Cではだめだから』です．Cでもオブジェクト指向のプログラムを書こうと思えば書けないことはないですが，それに1万行を超えるプログラムを

Cで書くほど能力もありませんし，そこで言語に頼るしかないというわけでC++を使うことにしたのです」.

いまでこそブームとなった感のあるC++だが，当時は日本はおろかアメリカでも知る人は決して多くはなかったという．同社でも最初はアメリカの製品を使っていたが，日本語が使えないうえに日本では十分なサポートも得られない．C++は使いたいがそんな処理系を使うのは不安もある．

「それならいっそのこと開発用の言語から自分たちで作ろうとAT&Tとライセンス契約を結んで最初はNEWS，次にMS-DOS用のC++を作りました．あるときふと世の中を見回してみるとほかには日本語で使えるC++がない．それならあるものは売ろうかということで売り出したわけです．もともとは一今でもそうですが一自分たちの開発用言語です」.

手続き型のプログラミング言語しか知らない人には難しいと言われるC++だが，同社ではC++でのプログラミングが常識だ．もちろんC++に興味を持った人が入社してはくるが，はじめからC++が書けるという人はほとんどいない．「入社前にC++を使っていた人はいませんね．うちは人数も少ないですし，決められた教育期間というのもありませんから，いきなりちょこちょことしたプログラムを書いてもらいます．それを見てここはもっとこうしたほうがいいとかアドバイスをするわけです．あとBBSに参加していればそこでいろいろと鍛えられますしね．そのうち非常に有名な人の書いた大きなソースプログラムを渡して，それを1か月くらいかけて解析してもらうんです．根性があれば（笑）それでなんとかなりますね」.

C++のサポートは続ける

このところC++のソースプログラムから直接実行形を生成できるC++ネイティブコンパイラがぞくぞくと登場している．トランスレータを販売するミワシステムズコンサルティングは，これらの新製品をどのように見ているのだろう．

「正直いってTurbo C++とMicrosoftのC++が出てくると（MIWA C++の販売は）つらいでしょう．Turbo C対応版のMIWA C++を出していますので，TURBO C++が出たときは全然売れなくなるんじゃないかと思っていました．それが意外にまだ出ているんですね．98をお使いでないユーザーの方，たとえばFM16βだとか．そういうちょっとメジャーではなくなってしまったハードウェア上で，しかも日本語を使いたいという方がまだまだいらっしゃる．そういうお客様がいる限りは，MIWA C++のサポートはこれからも続けていきます．

MicrosoftのC++については，まだ実際に見ていませんので何ともいえませんが，いずれはCのときのようにMicrosoftとBorlandの2社で寡占状態になるでしょうね．しかたないことだと思います．うちとしては，ワークステーションのほうにシフトしていくということも考えています．アーキテクチャもRISCが投入されてから面白くなってきましたしね．速い，いい環境が作れれば面白いと思います」.

今後はワークステーションへ

「UNIXのGUI，X-WINDOWの上でC++を使って本格的にアプリケ

About Company

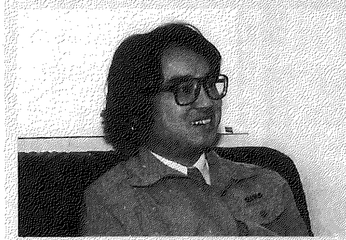
足立謙悟社長．1986年10月，ミワ電気工事株式会社からシステム開発室が独立して設立された．コンピュータによる各種業務の運用，管理に関するコンサルタント業務，システム分析，設計，プログラム開発，OA機器の販売，コンサルタントなどを行う．

Location

企画開発部は横浜駅西口から徒歩で7～8分の高台にある．にぎやかな駅前とは違って意外にも周囲は静か．

Profile

足立高德．学生時代は情報科学を専攻．早くからC++言語に注目してきた．C++言語の設計・開発者であるStroustrup博士とも親交があり，現在も手紙のやりとりを行っている．著書に「C++入門一次世代C言語によるプログラミングの実際」（CQ出版／1,550円）がある．また，日経MIXではawk会議の議長をつとめる．



Products

「電気工事積算システムMEE Ver.2.1」
もとはミワ電気工事株式会社の社内業務処理のために作られたシステムであったが，他社からの強い要望に応えて販売をはじめる．現在のバージョンは、同社のMIWA C++を用いて記述されている．

「MIWA C++」
日本語対応のC++トランスレータ．AT&Tの第2版に準拠したC++ソースプログラムを，対応するCコンパイラ用のソースプログラムに変換する．実行系の作成には別途Cコンパイラが必要．SONY NEWSシリーズ用，SONY RISC NEWSシリーズ用，NEC EWS-4800シリーズ用，日立 2050G

用など各社のワークステーションに対応．パソコン上ではMS-C（Ver.4.0および5.1），Turbo C（Ver.1.5および2.0）などのコンパイラに対応．PC-9800シリーズに限らず，富士通FMR-60やIBM5560，東芝J-3100シリーズ上でも動作が確認されている．

C++とは

Cにオブジェクト指向の概念を取り入れた言語で，AT&TのベルBjarne Stroustrup博士によって設計・開発された．最初は「クラス付きC」（C with Classes）の名で呼ばれていたが，その後拡張を重ねて現在はC++ 2.0となっている．データとそれを操作する手続きをパッケージ化して，所定の手続きによらないデータアクセスを拒否する「データの抽象化」機能を持つ．プログラムのカプセル化，部品化が可能となり，大規模なアプリケーション開発に適している．

Stroustrup博士の著書「THE C++ PROGRAMMING LANGUAGE」（邦題「プログラミング言語C++」斉藤信男訳／トッパン）はC++のバイブルと呼ばれている．

MIWA-net

ミワシステムズコンサルティングの運営するBBS．もともとはユーザーサポート用に開設されたが，MIWA C++のユーザーでなくてもC++に興味のある人の参加は歓迎している．日本でのC++関係の書籍の翻訳者など，C++に詳しい人が多数参加しているという．（045-321-6757）

開発環境

・マシン

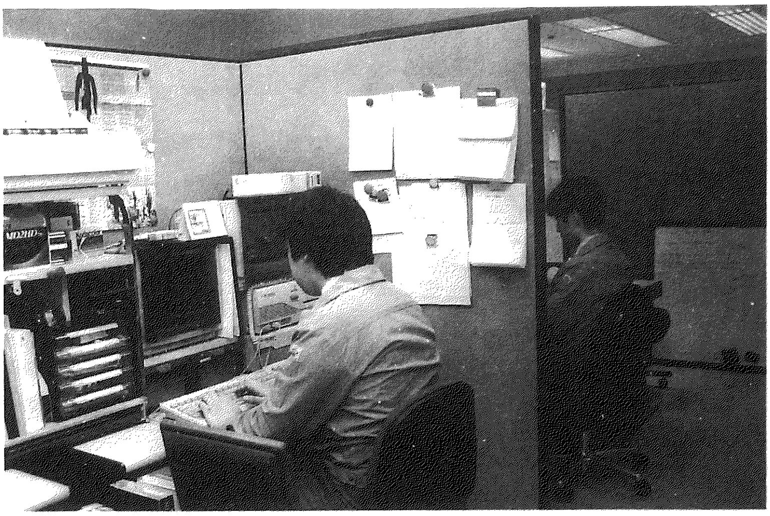
ワークステーション：SONY NEWS, SONY RISC, NEC EWS-4800, HITACHI 2050Gなど

パソコン：NEC PC-9801RA, EPSON PC-386LS, TOSHIBA J-3100SGT, Apple Machintosh SE/30など

・OSと開発言語

UNIX：MIWA C++ Ver.2.0, C

MS-DOS：MIWA C++ Ver.1.2, Microsoft C, Turbo C, MASM



ーションを書いているところはまだないと思います．そういう環境ーその環境そのものまで売り出すかどうかは別ですがーでアプリケーションを作っていきたいです．会社のステータスや人材募集という観点では，基本ソフトを開発しているというのはメリットになるわけですが，GNUのようなものを見てもわかるように，基本ソフト自体はだんだん空気のようなものになっていくと思うんです．基本ソフトだけで食べていくことはできないですから，やっぱりアプリケーションの開発ですね．アプリケーションはお使いになるお客さまに対しては手間がかかりますし，サービスでお金をいただくしかない．そのサービスを提供するためのしっかりした技術力さえあれば，将来も大丈夫だと思います．かっこいい技術を使っていてもそれをあまり外に見せない．スピードが速い，使い勝手がいい，そういう形で表せるような製品を作っていきたいです．そんななかで将来技術的に単体で売れるものが出てくれば，それも売っていくとは思いますが．ただ，製品として出すためにはかっこよくしてから出さなくてはならないのでたいへんなんですけど（笑）」．

C++を学ぶには?

最後に，C++のプログラミングを学ぶにはどうしたらいいかを足立氏にたずねてみた．

「Cを使ったことがあるという程度ではだめですね．Cを使いこなせますというくらいでないとC++は難しいでしょう．そのうえでパラダイムのシフトです．オブジェクト指向という考え方に慣れるまでに3か月くらいかかりますね．巷でいわれているほど簡単ではありませんよ（笑）．そんなに誰でもすぐにC++にシフトするというわけにはいかないでしょう」．

しかし，C言語のエキスパートでなくとも悲観することはない．

「私の本はCすらしらない人を対象にして書いています（ちょっと時期的には早すぎましたね）．オブジェクト指向という言葉を出さなくても，最初からそういう教育をすれば，何も知らない人にC++を教えることは可能だと思います．それにはほかの言語をご存じの方でしたら，Cを経由せずにC++に移ってくることはできると思います」とのことだ．

information・page

イベント

■4月17日(水)～4月19日(金)
EP'91
電子出版物やDTPなどの電子編集システムおよびマルチメディア関連製品を展示。
会場：池袋サンシャインシティユニバーサルホール／入場料：無料
問い合わせ先：日本経済新聞社事業局総合事業部　TEL 03-3252-8157

■4月17日(水)～4月20日(土)
EDUCATION JAPAN CAI & 教育メディア・ショー'91
学校および企業内教育向けCAIや教育メディアの展示とセミナー。
会場：池袋サンシャインシティ センターホールA・B／入場料：500円
問い合わせ先：日本経済新聞社事業局総合事業部　TEL 03-3252-8157

■5月8日(水)～5月11日(土)
マイクロコンピュータショウ'91
半導体・電子デバイスから応用機器・ソフトウェアシステムなどの展示とセミナー。
会場：東京流通センター／入場料：無料
問い合わせ先：（社）日本電子工業振興協会　TEL 03-3433-4547

■5月15日(水)～5月18日(土)
第72回ビジネスショウ
事務機、情報処理機器の総合展示会。
※15・16日は招待日、17・18日は一般公開日。
会場：東京国際見本市会場／入場料：無料
問い合わせ先：（社）日本経営協会　TEL 03-3403-1331

海外イベント

■5月20日(月)～5月23日(木)
COMDEX/Spring'91
コンピュータのディーラー向け展示会。WINDOWS WORLD'91も同時開催。
会場：米国アトランタ／入場料：無料
問い合わせ先：ザ・インターフェイス・グループ・インク日本支社　TEL 03-3237-8759

セミナー

Microsoft University

■4月22日(月)～4月26日(金)
MS OS/2 Application Programming
OS/2のAPIを使った実習中心のコース。
会場：富士ソフトウェアMUセミナールーム
受講料：150,000円（5日間）

■4月23日(火)～4月24日(水)、5月15日(水)～5月16日(木)
C言語入門
Quick Cを使ったC言語の入門者のためのコース。
会場：特別会場T
受講料：60,000円（2日間）

■5月20日(月)
Microsoft Windows入門
Windows 3.0の概略を理解するための入門コース。
会場：富士ソフトウェアMUセミナールーム
受講料：30,000円（1日間）

■5月21日(火)～5月24日(金)
MS OS/2 Programming Environment
OS/2アプリケーション開発のための基本コース。
会場：富士ソフトウェアMUセミナールーム
受講料：120,000円（4日間）

■6月5日(水)～6月7日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
会場：特別会場T
受講料：90,000円（2日間）

■5月7日(火)～5月10日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース。
会場：富士ソフトウェアMUセミナールーム
受講料：150,000円（4日間）

■5月13日(月)～5月17日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための基本コース。
会場：特別会場T
受講料：150,000円（5日間）

■5月13日(月)～5月17日(金)、5月27日(月)～5月31日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
会場：富士ソフトウェアMUセミナールーム、特別会場T
受講料：150,000円（5日間）

■5月27日(月)～5月31日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース。
会場：特別会場T
受講料：180,000円（5日間）

Microsoft Universityは、OS/2、Windowsアプリケーション開発者のために定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウェア(株)MU教育センター　TEL 03-3452-772

NEWS & NEW PRODUCTS

ボーランドジャパン，PC-9801版Turbo Pascal 6.0を発売

(株) ボーランドジャパン（TEL 03-3486-1400）は，PC-9800シリーズ用「TURBO PASCAL 6.0」および「TURBO PASCAL 6.0 PROFESSIONAL」を発売した．「TURBO PASCAL 6.0」には，オブジェクト指向によるユーザーインターフェースライブラリTurbo Visionが付属している．「TURBO PASCAL 6.0 PROFESSIONAL」には，プロテクトモードで動作するコマンドライン版コンパイラが付属している．

■おもな仕様

対応機種：PC-9800シリーズ（ハイレゾモード対応，LTを除く），EPSON PC-286/386シリーズ／環境：MS-DOS Ver.2.11以降，必要メモリ640KB以上，バスマウス対応，EMS対応，PROFESSIONALのプロテクトモードコンパイラ使用にはプロテクトメモリ1MB以上必要／価格：TURBO PASCAL 6.0 29,800円，同PROFESSIONAL 59,800円

マイクロソフト，MS-C 6.0用開発支援ツールパッケージを発売

マイクロソフト（株）（インフォメーションセンター：TEL 03-3363-1201）は，MS-C 6.0における開発環境を向上する3種類のツールを1つにパッケージした「Microsoft C Developer's Toolkit」を発売した．これには，「マイクロソフト C 6.0 ランタイム ライブラリ ソースコード」「マイクロソフト プロフェッショナル アドバイザー ライブラリ」「マイクロソフト C プログラマーズ ツールキット」の3種類のツールがパッケージされている．

■おもな仕様

環境：「Microsoft C Professional Development System Ver.6.0」および「Microsoft Macro Assembler Ver.5.1」以上，MS-DOS Ver.3.1以降・必要メモリ640KB以上またはMS OS/2 Ver.1.1A以降・必要メモリ1MB以上，3MB以上の空き領域のあるハードディスク／価格：50,000円

レンガ，C言語ファイル圧縮ライブラリを発売

(有) レンガ（TEL 03-3494-1098）は，MS-CおよびQuick C言語のファイル圧縮のためのライブラリ「LZSSシステム」を発売した．C言語アプリケーション開発にこのライブラリを使用することにより，テキストファイルで37%，バイナリファイルで75%，画像データで31%（最高13%）まで圧縮することができる．価格は，MS-DOS版が88,000円．

また，MS-C 6.0のアドバイザー機能に対応した，C言語アプリケーション開発用ライブラリ＆開発支援ユーティリティセット「C-RDB Ver.2.0 MS-C 6.0 Advisor対応版」（価格：68,000円）も発売．

オオツカ商事，C言語開発支援ツールCPAD Ver.2.0を発売

(有) オオツカ商事（ユーザーサポートセンター：TEL 0878-51-1807）は，「CPAD」の新バージョン「CPAD Ver.2.0」(PC-9800シリーズ版) を発売した．「CPAD」は，C言語による開発時の工数短縮および複数人数によるプロジェクト管理を容易にするC言語開発支援ツール．価格は，85,000円．

FAシステムエンジニアリング，C言語用開発支援ライブラリ2種を発売

FAシステムエンジニアリング（株）（TEL 0899-31-2886）は，C言語用開発支援ライブラリ「簡単シリーズ」に，「エディタ入力が簡単」「ヘルプが簡単」の2本を追加発売した．価格は，「エディタ入力が簡単」が22,000円，「ヘルプが簡単」が28,000円．

ワンラインNEWS
米HPとサン・マイクロ・システムズはWSなどのソフトの互換性を目指して提携した．狙いは市場拡大であるが，UNIXの標準化へも影響が出ると見られてる．

NEWS & NEW PRODUCTS

オーキッド，効率的な環境を構築するC言語用開発ツールを発売

(株) オーキッド (TEL 03-3827-7496) は，エディタ・シェル・MAKEの連携プレーにより，効率よくCプログラムを作成できる開発支援ツール「C Surround」を発売した．シェル，エディタ，MAKEなどの基本ツールに加え，メモリ情報の出力，EMS情報の出力，ファイル検索などのツールも含まれている．価格は，19,500円．

マイクロフォーカスジャパン，if386 AXなどに対応するCOBOL/2の供給を開始

マイクロフォーカスジャパン(株) (03-3486-7791)は，沖電気工業 (株) のif386 AXおよびif800 EXに対し，高機能最新コンパイラ「Micro Focus 日本語COBOL/2」および自在に画面設計ができる「Forms-2」のOEMライセンス供給を開始した．コンパイラにはデバッキング・ユーティリティ「ANIMATOR」が付属している．

エプソン，3.5インチFDD2基内蔵の32ビットマシンを発売

セイコーエプソン (株)，エプソン販売 (株) (エプソンPCインフォメーションセンター：TEL 03-3377-3531)は，3.5インチFDD 2 基搭載の32ビットデスクトップパソコン「エプソン PC-386S-UST」を発売した．この機種は，「PC-386S」の機能をそのままに，FDDを3.5インチタイプとしたもの．

■おもな仕様

プロセッサ：80386 (25MHz) ／メモリ：1.6MB／外部記憶装置：3.5インチFDD×2／拡張スロット×2 (内部) ／価格：498,000円



日本IBM，日本語Windows V3.0をインストールしたPS/55Zの新モデルを発売

日本アイ・ビー・エム (株) (TEL 03-3586-1111) は，「日本語Microsoft Windows V3.0」を低価格で最適に稼働できるPS/55Zの最上位モデル「5530-SW4システム装置」(Windows対応版) を発売した．現行では標準で2MBであった主記憶装置を6MBに拡張し，最大16MBまで拡張できる．システムに「DOS バージョンJ/4.0V」(簡易マニュアル版) と「日本語MS-Windows V3.0」，「連文節プログラム」をあらかじめ導入してあり，購入後すぐに利用できるようになっている．

■おもな仕様

プロセッサ：80386SX (16MHz) ／メモリ：6MB／外部記憶装置：3.5インチFDD×1・40MB HDD×1／価格：671,000円

日本IBM，DOS/Vに対応した初のノート型パソコンを発売

日本アイ・ビー・エム (株) (TEL 03-3586-1111) は，DOS/Vに対応した，32ビットノート型パソコン「PS/55note」を発表した．A4サイズで，重さ2.5キログラム，VGA搭載．ノート型パソコンとしては大型の9.5インチ16階調白黒液晶ディスプレイを採用し，漢字40字25行表示が可能．

ワンラインNEWS
東芝は小型HDDの駆動装置の生産体制を強化し，岐阜工場内に同装置の製造建屋を建設する．小型HDDはノート型パソコンなどで需要拡大が見込めるため．

NEWS & NEW PRODUCTS

■おもな仕様

プロセッサ：80386SX（12MHz）／メモリ：2MB（最大6MBまで拡張可能）／外部記憶装置：3.5インチFDD×1（S01・SJ1），外付3.5インチFDD×1・40MB HDD×1（S04・SJ4）／価格：232,000円（S01），248,000円（SJ1：DOS/V連文節変換プログラム付），352,000円（S04），368,000円（SJ4：DOS/V，連文節変換プログラムをHDDに導入）

富士ゼロックス，ワークステーションの新機種を発売

富士ゼロックス（株）（TEL 03-3585-6792）は，ローカルエリア・ネットワーク（LAN）システムの中核となる多機能ワークステーションとして「富士ゼロックス　8087-Ｊスター II」を発売した．この機種は，「Ｊスター」シリーズの最新モデルで，「8083-Ｊスター II」および「8085-Ｊスター II」の後継機となる．主メモリ8MB，ハードディスク90MBのタイプと主メモリ2MB，ハードディスク45MBのタイプがある．価格は，1,350,000円から1,980,000円．

日本エヌ・シー・アール，ワークステーションの新機種を発売

日本エヌ・シー・アール（株）（TEL 03-3582-6111）は，「NCR9005 II ／ II E」の後継機として，80486搭載の「NCR9005 III モデル4303」の下位機種にあたる「NCR9005 III モデル2302パーソナル・ワークステーション」を発売した．CPUに80386（20MHz）を搭載．高解像度ディスプレイ（1170×780ドット）を採用，メモリは標準で2MB，専用メモリスロットで最大8MBまで拡張可能．価格は，900,000円．

アップルコンピュータジャパン，レーザー品質のプリンタ2種を発売

アップルコンピュータジャパン（株）（TEL 03-5562-6215）は，従来より価格を大幅に引き下げた２種類のプリンタを発売した．「Apple StyleWrite」は，低価格でレーザー品質をもったインクジェット方式のプリンタで，価格は98,000円．「Apple Personal LaserWriter LS」は，同社で最も低価格のレーザープリンタで，価格は，248,000円．

富士通，全二重・半二重の通信が可能な複合モデムを発売

富士通（株）（ターミナルシステム販売推進部TEL 03-3216-3211）は，半二重モード，全二重モードに１台で対応できる電話回線用ボックス型複合モデム「富士通SA4824」を発売した．これにより，半二重でビジネスネットワークと全二重でパソコン通信などが可能になる．通信速度は，半二重で4800bps，全二重で2400bps，MNP5に対応している．価格は，280,000円．

日本IBM，PS/55用プリンタの新機種を発売

日本IBM（TEL 03-3586-1111）は，PS/55用の卓上型プリンタとして，「IBM5575-H02印刷装置」を発売した．24ドット・ワイヤー・シリアル方式のプリンタで，同価格の現行モデル「5575-B02」と比較して印字速度を約25％高速化した．価格は，265,000円．

エプソン，IBMパソコン対応のカラー漢字プリンタを発売

セイコーエプソン（株），エプソン販売（株）（エプソン販売：TEL 03-3377-7001）は，IBM PS/55，PS/55Z，4692POSおよび海外IBM PCシリーズ対応の24ピンドットマトリクスカラー漢字プリンタ「エプソン

ワンラインNEWS
アルプス電気はWindows 3.0でフォントを自由に伸縮できるアウトラインフォントの拡張ボードを開発．６月より発売する．

NEWS & NEW PRODUCTS

BM-2000」(136桁) と「エプソン　BM-900」(80桁) の2機種を発売した．これらは「エプソンBM-1000/800」の後継機種で，欧米語モードのサポート，カラー印字機構の搭載，シートフィーダの標準装備，印字のスピードアップ，32ビットマシン対応などの機能を強化．価格は，BM-2000が188,000円，BM-900が158,000円．

アイ・オー・データ機器，PC-9801DA用RAMボードを発売

アイ・オー・データ機器 (TEL 0762-21-4812) は，NEC PC-9801DA用RAMボード「PIO-DA134シリーズ」を発売した．容量は，4MBから12MBまでの5種類．価格は，49,800円(4MB)から145,000円(12MB)．

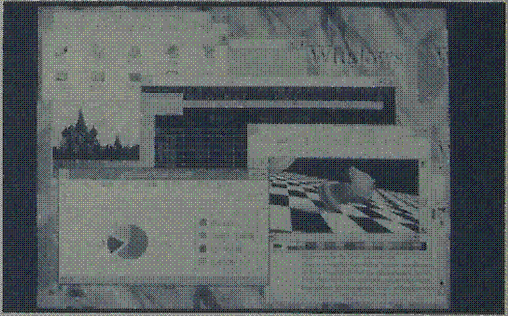
富士通，高精細表示ディスプレイを発売

富士通(株) (TEL 03-3215-2398) は，16インチの画面に，1,280×1,024ドットの表示容量および解像度100dpiの高精細表示を実現したプラズマディスプレイ「FPF20000S-001」を発売した．奥行き9cmの省スペース対応のスタンドアロン構造で，消費電力25Wの省電力を実現した．価格は，600,000円．

日本IBM，高解像度とVGAに対応した日本語Windows V3.0を発売

日本アイ・ビー・エム (株) (TEL 03-3586-1111) は，DOS/V上で動作する「日本語Microsoft Windows V3.0」を発売した．「日本語Microsoft Windows V3.0」は，「DOS ver.J4.0/V」で稼働するが，PS/55の高解像度ディスプレイアダプタの1024×480ドットをサポートしている．386拡張モードでは，VGA (640×480) のフルスクリーンをウィンドウ表示することができ，日本語・英語のDOS・Windowsアプリケーションをすべて動作させることができる．価格は，25,000円．また，PS/55ZにWindowsをインストールしたモデルも追加された．

このほかにも，日本語オペレーティング・システム／2 (OS/2) の強化版「OS/2基本版J1.3」と「OS/2拡張版J1.3」，PS/55ファミリーの中上位モデルに内蔵できる「3.5型光磁気ディスク駆動機構II/A」，価格性能比を向上したPS/55の新モデル「PS-55モデル5550-V24/V28/V2B」などを発売した．



アップルコンピュータジャパン，日本語版および英語版HyperCard 2.0を発売

アップルコンピュータジャパン(株) (TEL 03-5562-6215) は，「日本語版HyperCard 2.0」および「英語版HyperCard 2.0」を発売した．これは，すでに出荷されている「日本語HyperCard 2.0 Lite」にスタックウェア開発のために必要なサンプルスタックや資料を追加したもの．価格は，日本語版／英語版ともに19,800円．

日本サン・マイクロシステムズ，日本語OpenWindowsの本格出荷を開始

日本サン・マイクロシステムズ (株) (TEL 03-3221-7228) は，Sunワークステーション用のウィンドウ環境である「日本語OpenWindows」のエンドユーザー向け本格出荷を開始した．すでに，国内の87社が141ア

NEWS & NEW PRODUCTS

プリケーションに「日本語OpenWindows」のサポートを決めており，オムロン・ソフトウェア(株)，(株)ブリッジなどが対応アプリケーションの提供を開始した．価格は，58,000円．

シンフォニー，MS-Windows 3.0対応高機能チャートエディタを発売

(株) シンフォニー (TEL 03-3443-0291) は，Microsoft Windows 3.0対応プレゼンテーションチャート，フローチャート作成用アプリケーション「RFFlow ver.2.0J (日本語版)」を発売した．これは，「RFFlow ver.1.0J (日本語版)」をWindows 3.0に対応，機能アップしたバージョンアップ製品．価格は，48,000円．

ボーランドジャパン，RDBソフトPC-9801版「PARADOX 3.5」を発売

(株) ボーランドジャパン (TEL 03-3486-1400) は，PC-9801用リレーショナルデータベース「PARADOX 3.5」とその関連製品を発売した．価格は，168,000円．新発売キャンペーンとして，1991年9月30日までは97,000円の特別価格で販売する．ボーランド製品の正規登録ユーザー (4月10日までの) には，1991年6月末日までは39,800円の特別価格にて販売．関連製品として，「PARADOX 3.5マルチパック」(198,000円)，「PARADOX ENGINE」(98,000円)，「PARADOX SQL Link」(98,000円)の発売が予定されている．

富士通，NetWare386を発売．製品は10・40・100クライアント用の3種

富士通 (株) (TEL 03-3216-7980) は，ノベル (株) のパソコンLAN用オペレーティングシステムNetWareの日本語版「NetWare386 v3.1J」を発売した．あわせて，パソコンLAN専門スタッフによるシステム構築支援サービスや教育コースの開設など，パソコンLANのためのトータルサービス (有償) を開始した．価格は，480,000円(10クライアント用)，980,000円 (40クライアント用)，1,600,000円 (100クライアント用)．

パーソナルメディア，カード型DB「Let'sアイリス」UNIX版を発売

パーソナルメディア (株) (03-3490-0410) は，パソコン用カード型リレーショナルデータベース「Let'sアイリス」を，サン・マイクロシステムズ社のUNIXマシンSunワークステーション上に移植し「Let'sアイリス／Sun」として発売した．価格は，198,000円 (4ユーザー用)，498,00円 (12ユーザー用)．

サザンパシフィック，英文文章解析ツールの98対応版を発売

(株) サザンパシフィック (TEL 045-314-9514) は，米国Reference Software International社の英文文章解析ツール「GrammatikIV」のEPSON PC-286/386シリーズ，NEC 9801シリーズ対応版を発売した．価格は，19,800円．

日立製作所，超高密度磁気記録技術を開発．3.5"で10GBのHDDも可能に

(株) 日立製作所 (中央研究所技術管理室：TEL 0423-23-1111)は，小型大容量ディスクの実現を可能にする，2Gb/in^2 (3億ビット／cm^2) の超高密度磁気記録技術を開発した．これは，現状の磁気ディスクの

ワンラインNEWS
富士通は同社のスーパーコンピュータ「VPシリーズ」の追加モデルを発売した．OSにはSVR4.0準拠のUNIXを採用している．

NEWS & NEW PRODUCTS

約20倍の記録密度にあたり，この技術を3.5インチのハード磁気ディスクに応用すると，1台で10GBの記録容量が実現可能となる．

ソニー，従来の6倍の密度をもつ光信号記録再生技術を開発

ソニー（株）（TEL 03-3448-2200）は，光磁気ディスクにおいて，光の物理的波長の制約による再生限界を超越して，微小信号を再生することが可能な原理に基づく，光信号記録再生技術「アイリスター」を開発した．この技術を採用することにより，現行の半導体レーザーにおいて6倍，将来は，青色レーザーと組み合わせることにより，原理的に20倍以上の高密度光磁気ディスクの実現が可能となる．

マイクロソフトの「Pen Windows」に21社が採用の意向を発表

1991年2月21，22日，米国ワシントン州シアトルにて「Microsoft Pen Windows デベロッパーズ　コンファレンス」が開催され，21のコンピュータメーカーなどが「Micrsoft Pen Windows」の採用を発表した．Pen Windowsは，Windows 3.0を基本に，タブレットとペンベースコンピュータ用に開発された新しいGUI（グラフィカルユーザーインターフェース）で，手書き文字の認識やデータの加工を行うことができる．日本企業では，日本電気，富士通，東芝など11社が採用の意向を発表した．

「日本語MS LANマネージャ共通規約」が策定

LANマネージャを利用したパソコンが，どのメーカーの装置であってもLANを介して互いに通信できるようにするため，「日本語MS LANマネージャ共通規約」が策定された．APIをそのまま使うことなど，基本的な点で合意された．この規約を策定したメンバーは，日本アイ・ビー・エム，日本電気，富士通などのコンピュータメーカーとLANハードウェアベンダー18社．

富士通，複数チップを1つのパッケージに搭載する技術を開発

富士通（株）（TEL 03-3216-3211）は，従来のパッケージ（DIP，SOJ，ZIP，TSOP）の中に，複数個のチップを搭載したメモリ用パッケージを開発した．この技術は，複数個のチップを重ねて搭載することにより，従来と同様のパッケージの中に，2倍から4倍までの容量増を実現できる．具体的には，現在量産を目指す16Mビットを，従来パッケージのまま32～64ビットへ記録密度の向上が可能となる．

日本IBM，PCオープンアーキテクチャー推進協議会（OADG）を設立

日本アイ・ビー・エム（株）は，異なるメーカーが製造販売するパーソナルコンピュータ間でのソフトウェア共通利用基盤の整備と推進を目的として，「PCオープン・アーキテクチャー推進協議会（OADG）」を設立した．当面は，DOS/Vを中心にオープンアーキテクチャの推進をはかる．三菱電機，日立製作所，東芝など11社が同協議会への参加の前提となる「PCオープンアーキテクチャ推進企業登録」への意向表明をしている．

シグマシステムとX/Open，OSF，UIとの協調関係確立

（株）シグマシステムは，国際的なオープンシステム推進団体であるX/Openカンパニーリミテッド，UNIX インターナショナル（UI）およびオープン　ソフトウェア　ファンデーション（OSF）とそれぞれ，オープンシステムの確立に向けた活動を協調して推進するための協定を締結した．シグマシステムは，Σプロジェクトの普及をはかるために設立された会社だが，その普及がゆきづまるなかで国際規格への歩み寄りをしたという見方が一般的だ．

Programmer's Program

フルスクリーン型画面設計エディタをもつ画面入出力設計支援ツール

PPII（ピーピーツー）

画面設計を支援する豊富なユーティリティ

アプリケーションソフトウェアを開発する上で，画面入出力など，プログラムの基本部分となるユーザーインターフェースの開発はコンパイラの標準ライブラリを使うだけではかなりの労力と時間が費やされる．しかも，これがある程度完成しないと，アプリケーション本体の開発もなかなか進まない．そして，画面や入出力の設計の良し悪しで，アプリケーションプログラムの評価が決まるといっても過言ではない．

「PPII」は「いかに画面入出力処理を要求通りの仕様で，どうインターフェースを組むか」などの悩みを一挙に解決させる，数多くのユーティリティと，多機能なユーザーインターフェース関数群をもったC言語用ツールである．

各ディスクごとにINSTALL.BATを実行する

「PPII」のパッケージには，2HDのディスク5枚，マニュアル5冊，ユーザー登録カード，サポート依頼書，そして「PPIIユーザーの方へ」と記したドキュメントが入っている（5インチ2HDの場合）．

PPIIの試用にはNECのPC-98RL model2を使った．Microsoft C Professional Development System 6.0（以下MS-C 6.0）がインストールされている100MBの外部HDDと4MBのEMSメモリが接続されている．PPIIには，Lattice C対応版とMS-C対応版があるが，今回はMS-C対応版を選択した．

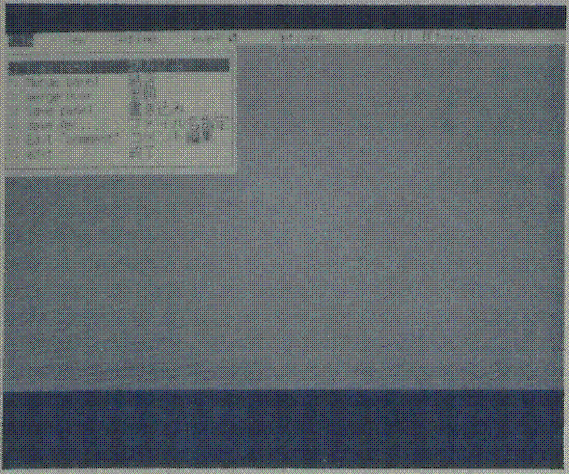
まず，マスターディスクのバックアップをとり，次にインストール作業に入る．インストールは各ディスクのルートにあるINSTALL.BATを実行する．たとえば，CドライブにPPIIをインストールする場合は，カレントがPPIIのマスターディスクが入ったディスクであることを確認して，DOSのコマンドラインから

```
A> INSTALL C:
```

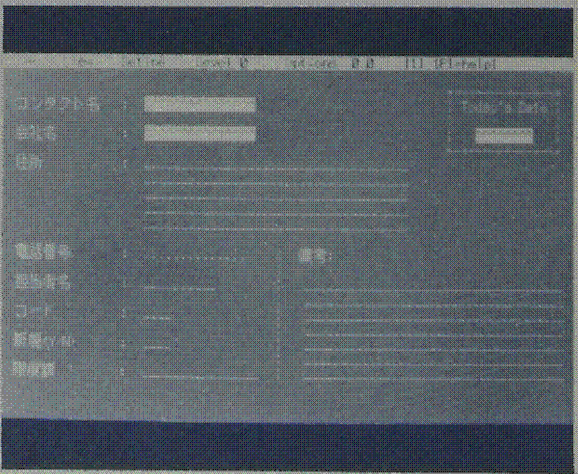
と入力し，[RETURN] キーを押す．これを各ディスクごとに実行すれば，インストール作業は完了する．

次は環境変数の設定である．PPIIのライブラリを使うには環境変数includeにPPIIのライブラリヘッダファイルのあるディレクトリを追加する．また「PPE」や「PANGENC」

■PPEの起動画面



■画面設計のサンプル



のあるディレクトリにパスを通す．ほかにもいくつかの環境変数があるが，これらについてはマニュアルを参照して欲しい．そして，PPIIのディレクトリにある環境ファイルPANELP.CNFで画面関連の設定を行い，準備が整ったことになる．

フルスクリーン型エディタで手軽にレイアウト

画面イメージを簡単にレイアウトできるフルスクリーン型画面制御エディタ「PPE」をPPIIは備えている．PPEの起動はDOSのコマンドラインから

```
A> PPE
```

と入力し，［RETURN］キーを押す．PPEの起動画面はメニューバーのみの表示で，いたってシンプルである．各種操作はマウス，またはキーボードによって行う．また操作でわからないことがあれば，［HELP］キーを押すことによって，いつでもオンラインヘルプを呼び出すことができる．

フルスクリーン上でマウスやカーソルキーを使って自在に位置を決め，ポップアップメニュー，階層的メニュー，仮想画面を使って上下左右にスクロールが可能なスクロールウィンドウ，ウィンドウの中でフィールドを重ね合わせるマルチオーバーラップウィンドウなどを選択し，フィールドと呼ばれる各関数で使用するデータ入出力や仮想画面，メニューなどのウィンドウを処理するためのデータ構造の定義を行うだけである．

関数や，各フィールドの属性（色や入力チェックなど）の指定もできる．

画面データをCソースコードに変換

このようにして，PNLという拡張子をもった画面データファイルができあがる．この画面データファイルをプログラムの実行時に読み込むことによって，PPIIのさまざまな関数から利用することができる．

さらに，この画面データをCソースコードジェネレータ「PANGENC」によって実行形式ファイル内に取り込むことが可能である．PANGENCは画面データファイルを読み込んでCのソースコードやヘッダファイルを出力するので，あとはアプリケーションプログラムとリンクするだけである．

◆

マニュアルは5冊で726ページとかなりのボリュームがあり，またチュートリアルプログラム（学習ソフト）も付属している．そして，なによりサンプルプログラムのソースが豊富に公開されているので，PPIIを使いこなすのに時間はたいして必要ないだろう．

また，PPIIの全関数を，エディタ「RED2」のオンラインガイダンス機能で参照できる，「ガイダンス・ファイル」を標準装備している．

ただし，C言語の基礎知識については触れられていないので，Cについては十分に理解している必要がある．

●PPII

販売：㈱ライフボート／価格：98,000円／動作環境：PC-98シリーズ（LT, XA, XL・XL2・RLのハイレゾモードを除く）／必要メモリ：640KB／動作OS：MS-DOS Ver.3.1,3.3／対応言語：Lattice C/DOS Ver.4.1, Lattic C/DUAL Ver.4.1, Microsoft C Ver.5.1, 6.0

【特徴】

豊富な画面制御機能（ポップアップメニュー，階層メニュー，仮想画面を使ったスクロールウィンドウ，ファイル選択ウィンドウ，入力チェック，キー入力待機中のバックグランド処理）／ライブラリ関数の完全ソースコードを添付／画面設計エディタ「PPE」／Cソースコードジェネレータ「PANGENC」／さまざまな目的に対応したユーティリティプログラムを付属／フィールドを重ね合わせるマルチオーバーラップウィンドウをサポート／いかなる場合でもヘルプ情報を表示できるオーバーラップ型ヘルプボックス機能／9段階のカリキュラムによる学習プログラム／数多くのサンプルプログラム／PP IIの全関数をエディタ「RED2」のオンラインガイダンス機能で参照できる「ガイダンスファイル」

【仕様】

最大フィールド数：制限なし／最大フィールドサイズ：縦・横255字（スクロール可能）／最大オーバーラップウィンドウ数：127個／最大表示色：128パターン（任意のタイルパターンを指定可能）／対応メモリモデル：全メモリモデルに対応



オンラインソフト採集帳 5

オリジナリティあふれる懐かしのビデオゲーム NUKEM

あらかじめマウスドライバを組み込む

「NUKEM」はマウスを使ったシューティングゲームで，ATARI社のビデオゲーム「ミサイルコマンド」とほぼ同じである．10年ほど前に登場した，当時としては奇異だったトラックボールを使い，派手な原色カラーを使ったオリジナリティあふれるゲームであった．最近のビデオゲームは画像こそ細かく，そしてきれいになったが，ゲーム自体のおもしろさは，どれも同じような内容で新鮮味に欠ける．

このゲームで用意する必要があるのはマウスだけである．このゲームはキーボードのみでもゲームはできるが，やはりマウスがないとゲームにならないであろう．また，マウスを使用するには，あらかじめマウスドライバが組み込まれている必要がある．マウスドライバはMS-DOSのシステムディスクに付属するドライバはもちろん，マイクロソフト社のドライバや，速水祐氏の作成したフリーウェアの「MSZ」にも対応．また，FM音源実装の機種であれば効果音を楽しめる．

起動オプションでマウスやサウンドを設定

NUKEMの起動はMS-DOSのコマンドラインから

```
A〉NUKEM -N -S
```

などと入力し，[RETURN]キーを押す．起動オプションでマウスやサウンドボードの設定をする．-Kはマウスドライバが組み込まれているときでもキーボードで操作する場合に指定．-NはMS-DOSのシステムディスクに付属のマウスドライバを，-Mはマイクロソフトのマウスドライバや MSZを使用する場合に指定．-Nや-Mの後ろに1から16まで数値を指定した場合，マウスの感度をその値に設定．-Lはミサイルの照準の色を白にして，モノクロディスプレイや液晶ディスプレイでゲームするときに見やすくする．-SはFM音源ボードに登載されている機種でEMSを使用していてサウンドBIOSのROMが無効になっている場合や，純正以外のFM音源ボードを使用していて音がおかしい場合にサウンドBIOSを使用せず，直接FM音源ICのレジスタを操作する場合の指定である．-Hや-?で簡単なオプションの説明を表示する．

起動するとデモンストレーションがはじまる．ここで，なにかキーを押せばゲーム開始である．またデモンストレーション中に[ESC]キーを押せば，ゲームを終了してMS-DOSのコマンドラインに戻る．

敵のミサイル攻撃から6つの都市を守る

ゲームは迎撃ミサイルを使って，空から降り注ぐ敵のミサイルを迎撃し，敵の攻撃から6つの都市を守ることである．すべての都市が破壊されたらゲームオーバーである．

マウスの移動で迎撃ミサイルの照準を合わせ，[S]・[D]・[F]または[L]・[;]・[:]の3つのキーで迎撃ミサイルの発射を行う．マウスがない場合は，迎撃ミサイルの照準は[↑]・[↓]・[←]・[→]キーや，テンキーの[2]・[4]・[6]・[8]で移動させる．同時に[SHIFT]キーか，[SPACE]キーを押せば，照準が倍速で移動する．

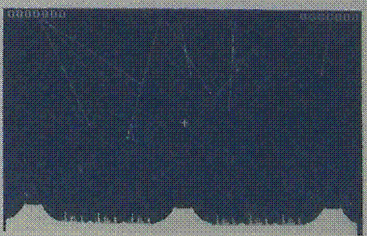
またマウスのクリックでゲームをはじめた場合には，マウスだけのプレイができるようになる．オリジナルのゲームでは迎撃ミサイルの基地は3つであるが，マウスには2個のボタンしかないので2つになる．

低高度で分裂する多弾頭ミサイルに注意

敵の攻撃は通常の多弾頭ミサイルのほかに，爆撃機から発射される多弾頭ミサイルがある．このミサイルは低高度で分裂する．ほかに自動回避型のミサイルがある．これは迎撃ミサイルの爆発を探知し，回避運動を行う．このミサイルは確実に都市や迎撃ミサイル基地に着弾する．これらのミサイルには注意が必要である．

敵のミサイルの速度が速くなってくると，いちいち照準を合わせていては間に合わなくなってくる．その時は開始と同時に迎撃ミサイルの弾幕を張る戦法が有効である．このとき，弾幕を張る高度と密度が重要になる．また，ラウンドが進んで敵の攻撃が激しくなってくると，すべての敵のすべての攻撃を迎撃するのは無理である．敵のミサイルの落下地点を迅速に見極め，無事な都市の上空を集中防御し，関係の無いところに落ちるミサイルは無視をすればよい．

このゲームは200ラインモードで8



ワンラインNEWS
日本IBMは3月から業界に先がけて3.5インチ光磁気ディスクを出荷すると発表した．同社は昨年より規格の標準化をすすめていたが，他社も追従する考え．

色のグラフィック画面だけ使用しているので，すべての98で稼動可能である．時間調整用にVSYNC割り込みが使用されているため，他のVSYNC割り込みを使用している常駐プログラムの動作がおかしくなる可能性があるので注意が必要である．

最近，次々と懐かしいゲームがリメイクされているのは嬉しいかぎりである．

■NUKEM

分類：ゲーム／登録名：NUKEM112.LZH／著作権者：井上健／動作環境：PC-9801シリーズ／入手場所：Genesis（03-3280-4571 24H ～9600 bps（V.32/HST））／解凍方法：LHA E NUKEM112

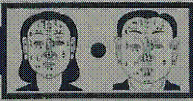
ファイル構成

<NUKEM112.LZH>

Name	Original	Packed	Ratio	Date	Time	Attr	Type	CRC
CHANGES.DOC	1126	690	61.3%	90-12-27	01:47:58	a--w	-lh1-	5CAD
NUKEM.EXE	61134	30611	50.1%	90-12-27	01:37:16	a--w	-lh1-	FC20
NUKEM112.DOC	8532	3987	46.7%	90-01-07	00:44:50	a--w	-lh1-	3205
3 files	70792	35288	49.8%	91-01-12	21:36:10			

★ VERSION UP INFOMATION ★

「FD」（2月号で紹介）のバージョンが2.30Cに（2月20日）
「LHa」（4月号で紹介）が「LHA」に名称を変更，バージョンが2.11に（3月3日）
「WTERM」（3月号で紹介）のバージョンが0.30に（3月13日）

HEALTHY?  UNHEALTHY?

⑤ 機能性食品はほんとうに機能する？

オリゴ糖やビフィズス菌などが一般に注目されるようになったのは，機能性飲料がブームを引き起こしてからのことだった．最近では，飲料類から機能性食品へと業界のターゲットが移っているようだ．

機能性食品には「食品」と「医薬品」の区別がある．たとえば，最近注目を浴びているものにエイコサペンタエン酸（EPA）という魚油の一種がある．これはコレステロールを下げ，動脈硬化を予防するとされている．EPAを軟カプセルにいれ，閉塞性動脈硬化症に使う場合は医薬品．またEPAを含む魚油をカプセルにいれたものが健康食品として売られている．医薬品の場合は公的な認可が必要なことはいうまでもない．

機能性食品は，EPAなどの成分を抽出して菓子や飲物にして，おいしくたべて病気を予防しようというものだ．「カロリーメイト」や「ポケメシ」につづいて，これからはさまざまな機能性食品が登場しそうだ．もっとも，食物成分を効率的に配合するという考え方は，家畜の飼料で以前から採用されていたもので，食品のエサ化がいっそう進んだという見方もできる．

■食物繊維

センイとかダイエタリーファイバーなどと呼ばれる．これには，セルロースのようにすじっぽい不溶性のものと，コンニャクのマンナンのようにピロピロした水溶性のものがある．食物繊維は，腸を刺激して便の排泄を促進したり，腸内の有害物質を体外に運びだす働きがある．

■ビフィズス菌

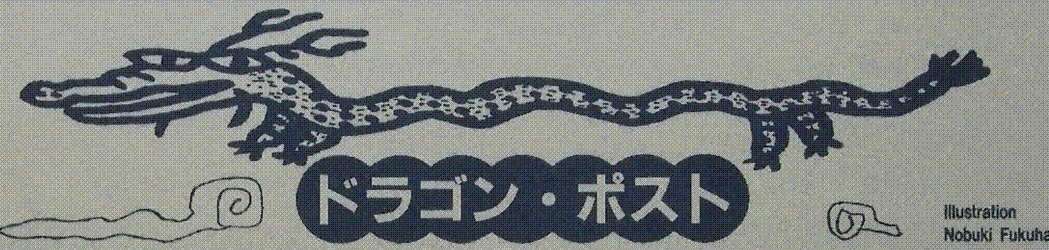
人間の腸内にはいろいろな細菌が住んでおり，消化やビタミンの合成，外部からの細菌による感染の防止などの働きをしている．ビフィズス菌はその中の乳酸菌の一種．腸内の細菌のバランスが崩れ，おなかの異常や下痢が起こるのを予防する．また発ガン性物質などの有害物質を生産する細菌の発育を抑える働きがある．

■オリゴ糖（フラクトオリゴ糖）

砂糖の30%ほどの甘さがあるが，消化管でほとんど吸収されないため低カロリーなのが魅力．甘味料として幅広く利用．また腸内の有用菌であるビフィズス菌を増やす．

■EPA（エイコサペンタエン酸）

イワシ，アジ，サンマなどの青魚の新鮮な血合いに多く含まれている脂肪酸で，脳梗塞や心筋梗塞の予防となる．コレステロールを低下させ，血栓の原因である血小板の凝集能力をさげる．



ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成します．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★33歳の過労死

2月22日付の新聞で「33歳のソフト会社員が過労死」というような内容の記事が掲載された．我が友も昨年とらばーゆしてソフトの会社に入ったが，フレックスタイム制とはいえ，ほとんどのように0時を超えるという．プログラマやソフト会社の皆さん体に気をつけて下さい．（橋本純一）

――と，おっしゃる橋本さんは，「過労死」の世界とは関係ないのでしょうね……．

★情報処理試験にモノ申す!

なぜに情報処理試験ではああも役に立たない問題を出題するでしょうか？　ハッキリ言って，関連知識の学習は人生の浪費である!　このように考えるのは私だけではありません．私の周りもみんな似たようなことをいいます．今度，実務と情報処理試験のギャップについて記事を組んではいかがでしょう．（元号もなくして欲しい）

★業界用語はむずかしい

愛読者カードの表の職業欄のことなんですけれど，ソフト技術者と技術系会社員はどう違うのでしょう．ちなみに私はCOBOLでプログラムを組んだりしているものなのですが，ところで「プログラマ」が意味するところも所によって違っているみたいで，どーもつかみどころのない業界だなと思う今日この頃です．（業界歴4月で1年）

――パソコンが趣味の分野から実務にも使われるようになってからそれほど年月を経ていません．汎用機の世界で確立した情報処理試験が現実と離れていくということも否めない面があります．

★マックで走るMindが欲しい

Mindのことは今までに聞いたことはあったのですが，これほど日本語だけでプログラムができるとは知りませんでした．マッキントッシュのHyperCardより，初心者にプログラムを教える時にアルゴリズムをふまえて教えられそうですね．マックで走るMindが欲しいなぁ～．（小林精一）

――「日本語でちゃんと組めるんだ」というのが，Mindを知った多くの人の感想であるようです．

★ソフト開発を主体とした雑誌に期待

最近のパソコン雑誌はハード，ソフトの体験とか，その使い勝手の案内ばかりで，パソコン雑誌登場時頃のソフトを主体とした雑誌が次々と廃刊になり残念に思っていた．ソフトクライシスといわれる割には，ソフト開発を主体とした雑誌がすたれるのに，やりきれない思いであった．たしかに，ハードでは欧米先進国を追い越したが，ハードのみでソフトなければただの箱であるように，ソフトなければ何の役にも立たない．最高のソフトが1つ出現するには，その下にソフト人口のいわゆるすその広がりが必要である．今回の貴誌の発刊は，このためにもおおいに期待したい．日本のソフトのためにも頑張ってほしく思っているところです．（加藤雄治）

――ひとくちにソフトといっても，とても幅広くて……．どこに焦点をあわせるか，雑誌づくりの悩ましいところでもあります．

★Cだけがプログラム言語じゃない

C言語全盛のご時世ですが，Cを途中で挫折した私にとっては，多くの雑誌がプログラム例をCで載せているのは少し苦々しい思いです．でも3月号で多くの言語を知り，励みになっています．CだけがPROの言語じゃないと，今，PascalかModula-2をやってみようと思っています．その関係もあるので，プログラムを載せるときアルゴリズム&流れ図も載せてくれると嬉しいのですが……．（狂夢）

――そうですね．実際のソフトウェア開発では，さまざまな言語が使われています．本誌は，Cの専門誌ではありませんので，実状に沿った企画を……ということで，Cの記事もけっこう多くなってしまいます．:-)

★誰にでも似てる？　表紙のイラスト

表紙のイラスト，みんな自分に似てると思っていませんか？　私にも似ています．私は誰にも似てないけれど，誰でも私にそっくりだ!（田中　晃）

★付録のフロッピーが5インチでは……

とうとう(!?)表紙のイラストが98からマックにかわりましたね．「なぜ98」という意見が多かったのですか．Townsはいつ表紙になるだろう．来月号よりフロッピーがつくそうですネ．でも5インチでは……．早く5インチディスクを増設しなくては．（高橋正巳）

――表紙のイラストは，『ビッグコミック』の表紙などでおなじみの日暮修一さんです．実は，表紙の人物にはちゃんとモデルがいます．劇団昴の桜井久直さんで，「先輩!甲子園じゃないんですから」というCMをやっていた方です．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

C平　このあいだ読んだ小説，すごく嘘がうまいの．普通は歴史上の出来事をモデルにしたり，どこかにいそうな人間を登場させて小説にするじゃない．これは，さも歴史書なんかに取材して書いているふりをしているけど，結局は全部嘘なんだ．

P子　なになに，『後宮小説』だって．後宮ってハレムみたいなもんでしょう．なんか，隠微じゃない．

C平　後宮や宦官が登場するから，中国王朝の話のように見えるんだ．実際，作者は中国の歴史についてすごく知識があるから，まことしやかな物語になっているだろうね．それに，説明の部分は歴史の講義風に書いてあるんだけど，登場人物の会話はまるで現代風だから楽に読めてしまう．もっともらしく書いてあるから，中国の歴史地図を調べちゃったよ．

P子　それって，ちょっとシフトした「カノッサの屈辱」じゃない．

C平　そういえば，歴史の講義風にもっともらしくっていうスタイルは同じだよね．ヒロインが銀河っていう名前だったり，皇帝がコリューンっていったり，今風なところもある．

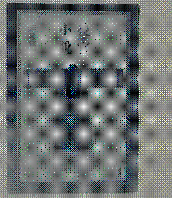
P子　あ，それって知ってる．アニメでやってた．うーん，『雲のように　風のように』っていうタイトルで，銀河っていうキャラクタがかわいかったのと，銀河の声を佐野量子がやったのをおぼえている．

C平　げっ！　アニメになってたの．そういえばアニメっぽい絵が帯についてたような……．

■後宮小説

酒見賢一
新潮社

第１回日本ファンタジーノベル大賞を受賞した「仮想歴史小説」．中国を思わせる国，素乾の先帝の腹上死により，新しい後宮をおこすところから物語ははじまる．宮女に応募した快活な少女，銀河はやがて皇帝コリューンの正妃となるのだが，フェミニズムを感じさせる後宮での宮女教育，三国志のような動乱など，多彩な味付けがされている．後宮という言葉から感じられる隠微さはなく，全編を貫く乾いた軽さが心地よい．



1,200円

読者プレゼント

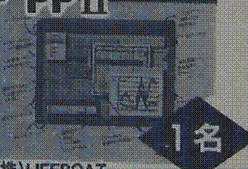
応募は本誌に添付のアンケートはがきをご利用ください（1991年５月17日消印有効）．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

1　C言語開発支援ツール　CPAD



1名
提供：(有)オオツカ商事

2　画面入出力設計支援ツール，ライブラリ　PPⅡ



1名
提供：(株)LIFEBOAT

3　CD「PacificHigh」　Philippe Kahn



10名
提供（株)ボーランドジャパン

4　プログラマのためのPCソースブック



1名

付録ディスクのご案内

このフロッピーディスクは5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクには，次のようなプログラムおよびデータを納めてあります．本誌およびドキュメントファイルをよく読んでからご利用ください．なお，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

●「ディスク目次」テキストファイル

ディレクトリ名：ルート（¥）

収録ファイル名

README.DOC（テキストファイル）

●特集・基礎力としてのアセンブラ

①いまどきのアセンブラプログラミング入門（p.4）

ディレクトリ名：TOKU1

収録ファイル名

LIST4.ASM	LIST7.ASM
LIST5.C	LIST8.ASM
LIST6.ASM	LIST9.ASM

③高級言語とアセンブラのプログラミング（p.37）

ディレクトリ名：TOKU3

収録ファイル名

A.ASM	BCTC.ASM
ASMFUNC.C	CALLC.ASM
B.ASM	CALLTEST.C
BC.C	LIST11.C
BCASM.ASM	LIST12.C
BCFUNC.C	LIST13.C
BCLSIC.ASM	LIST14.C
BCMSC.ASM	LIST17.C

④逆アセンブルの楽しみ（p.51）

MDEV450.LZH，およびSGEN131.LZHはオンラインソフトです．付属のドキュメントファイルに使用条件などが記されています．

ディレクトリ名：TOKU4

収録ファイル名

KZAT.COM	KZAT_SG.ENT
KZAT_AG.ASM	KZAT_SR.ASM
KZAT_AG.SEQ	KZAT_SR.DEF
KZAT_MDE.LST	KZAT_SR.LST
KZAT_ORG.ASM	KZAT_ZZ.ASM
KZAT_RIN.ASM	MDEV450.LZH
KZAT_SG.ASM	SGEN131.LZH

●MS-DOSツール・ラボ（p.78）

割り込みベクタ表示設定プログラム「VECT」

ディレクトリ名：TOOL

収録ファイル名

CONSOLE.ASM	SETPARAM.ASM
CONSOLE.H	VECT.ASM
DOSFUNC.H	VECT.EXE

●98活用アセンブラ技法（p.93）

ディレクトリ名：98ASM

収録ファイル名

BACKCOL.ASM

●プログラム『遊』工房（p.72）

パズル型ゲーム「mine sweeper」

ディレクトリ名：PLAY

収録ファイル名

MSWEEPER.EXE	LIST3.C
MSWEEPER.C	LIST4.C
LIST2.C	

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名

8086ASM.TXT（テキストファイル）

●オンラインソフト採集帳（p.138）

オンラインソフト採集帳の「NUKEM」，および4月号で紹介した高圧縮復元書庫管理ソフト「LHA」が公式公開版になったので再収録します．LHAは自己復元ファイルなので，MS-DOSのコマンドラインから実行するだけでファイルが復元されます．

ディレクトリ名：ONLINE

収録ファイル名

NUKEM112.LZH「NUKEM」

LHA211.EXE　「LHA」

●クイズ・ランダムスクリーン

クイズ解読ツール「PUZLE」

本誌掲載のクイズランダムスクリーン（p.143）の画像をディスプレイ上に再現し，マウスを使って並び替えることができるユーティリティです．対応機種はPC-9801シリーズです．

ディレクトリ名：OMAKE

収録ファイル名

5月号の問題

PUZLE.EXE	RNDSCREN.B1
PUZLE.DOC	RNDSCREN.E1
RNDSCREN.R1	RNDSCREN.RGB
RNDSCREN.G1	

4月号の解答

KAMOME.R1	KAMOME.E1
KAMOME.G1	KAMOME.RGB
KAMOME.B1	

Windows 3.0用 WallPaper（壁紙）

KAMOME.BMP

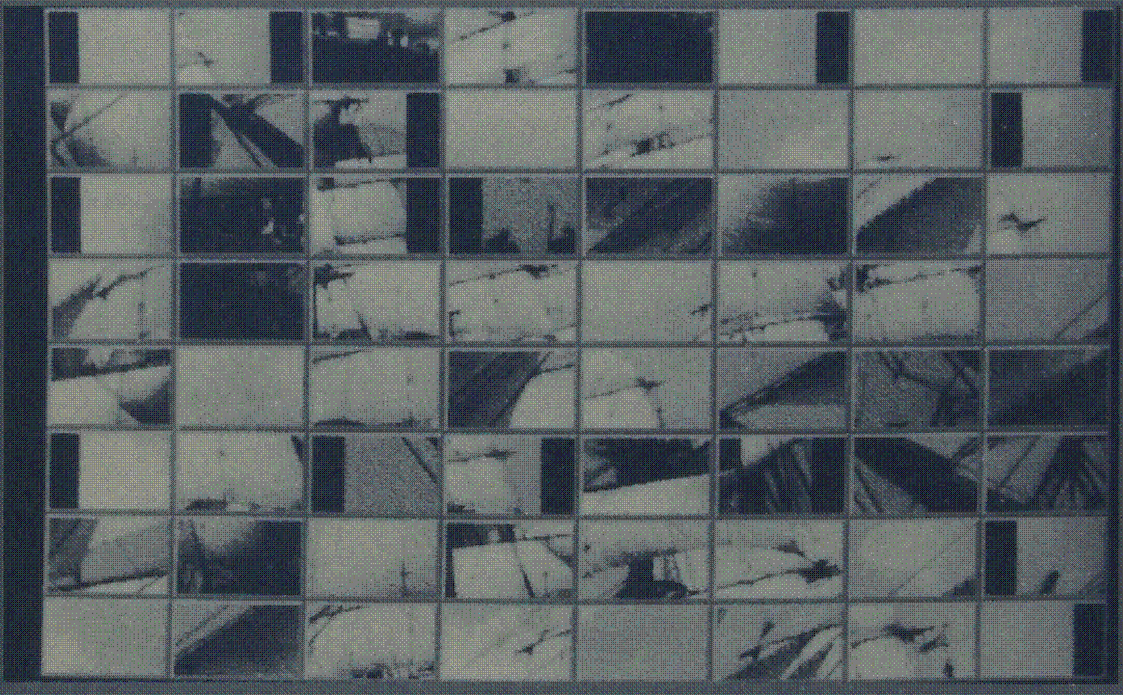
＊使用上の注意

本誌掲載プログラムおよびデータを利用するにはMS-DOSの基礎知識が必要です．また，このディスクにはMS-DOSのシステムは入っていません．別途MS-DOSが必要です．

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：横浜の名所．答えは○○でも○○○でもOK……



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

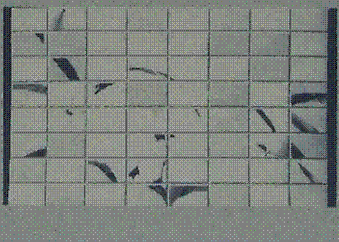
締め切り：1991年５月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

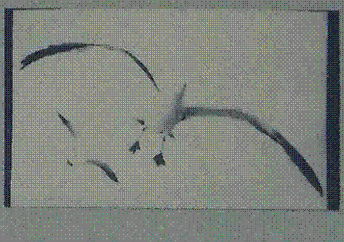
＊画像データは，付録ディスクやBBSでもサービスしています．

4月号の問題と正解

【問題】



【正解：カモメ】



月刊プログラマーズ・ページ オンラインサービス（SE-NET）のお知らせ

付録にフロッピーディスクをつけたため，ディスクによるプログラムサービスは中止いたします．オンラインサービスのBBSは，多数の読者の登録をいただいておりますので，今後も継続いたします．編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

①接続するとIDの入力を求めてきますので，GUESTと入力しリターンキーを押してください．
②お知らせを表示後，メインメニューのコマンドプロンプトが表示されます．
③月刊プログラマーズ・ページへのご意見などは「ドラゴンポスト」ボードに書き込んでください．
④ファイルの転送が終了したら，0と入力して回線を切ります．

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ6月号

(5月18日発売)

付録：5″ 2HDフロッピーディスク付き!!

特集
プログラミング統合環境

エディタ，コンパイラ，デバッガなど，いろいろなプログラミングツールを同じプラットホームで使うことのできる開発環境が一般的なものになっている．「統合環境」はプログラミングをどう変えたのか？何をめざしているのか？MS-DOS，OS/2，Mac，それぞれの立場から徹底検証する．

月刊プログラマーズ・ページ　1991年5月号
1991年5月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘＋勝山まゆみ
デザインスタッフ　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋石崎健治
協力　（株）ドモン・マインズ＋佐藤裕美＋小林弘人（図版デザイン）
写植　株式会社東巧社＋株式会社トーヨー・プリント社＋株式会社帆風
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

●バックナンバーのお知らせ
本誌バックナンバーに関するお問い合わせが増えています．バックナンバーは，書店にご注文いただければ入手できますが，内容，常備店などについては，次号よりお知らせいたします．

EDITOR'S PAGE

●パソコンに納められて出荷されるCPUは，総数からみるとほんの一部でしかないという．ファジー制御の家電製品に，自動車に，飛行機に……たくさんのCPUが使われている．それらもプログラマが組み上げたプログラムで動作しているはずだ．機会があれば，もうひとつのプログラムの世界を探ってみたいものだ．(TA)

●ついにボーランドがWindowsに対応した処理系を発売した．Borland C++，パッケージにはActorを開発したWhitewater社のResource Toolkitが含まれ，SDKは不要．価格がMS-Cとほぼ同額になってしまったのは残念だが，SDKが含まれていると思えばMS-Cの半額というところか．本誌2月号で取り上げたWindowsMAKERにもProfessional版が現れ，Smalltalk V/Windowsも発売された．また，Windowsプログラミングがおもしろくなってきた．(嘉平)

●冷たい雨の降るある日のこと，ふと立ち寄ったT-ZONEにはなぜかPC98用のWindows3.0のパッケージが山積みに．「品不足なんてやっぱりデマね，でも重たいからヨソで買おう」と考えた私が甘かった！駅近くのお店ではどこもみんな品切れ，Windowsの影も形もなかったのでした．思えば私の秋葉原散策も長いことラジ館どまりだったものです．それがいまでは銀座線の末広町，昌平橋のロケット，神田やぶそば，日比谷線の秋葉原駅を結ぶラインまで縄張りを広げるにいたりました．欲しいものを手にいれるにはマメに見て歩くこと，そしてやっぱり自分の足ですね．(kos)

●マウイ島に鯨を見に行った．カメラやビデオを片手にはしゃいでいる観光客満載のボートに平気で近寄づいてくる，人なつっこさに驚いた．実はこれ，反対に鯨が観光客をウォッチングしているのだそうだ．鯨の目にはどんなふうに映っていることやら．(ま)

●4月といえば，桜・進学・入社・昇進・異動などの季節．というわけで，私もこの度P.P編集部から他の編集部に異動することになった．創刊以来わずか半年の短い間だったが，すでに黄金の（？）歴史は綴られている．いつの間にかフィッと消えてしまった某連載，知らぬうちにはじまっていた新コーナー……．これからもP.Pは，コンピュータの世界のスピードと読者のワガママによって，その姿を変えていくことだろう．(K)

●付属のディスクの締切日，僕は二日酔いで会社を休んだ．前の週にディスクは作っておいたので，安心して自宅で死んでいた．そのころ，会社ではこのディスクにいくつか問題が発覚し，慌てたらしい．そのままデュプリされていたら，読者の皆さんに迷惑を掛けるところだった．(よ)

お詫びと訂正　本誌3月号記事中に誤りがありました．ここに訂正すると共に関係各位にお詫び申し上げます．
58ページ，誤：Lambdd Software Works　正：Lambda Software Works
60ページ，誤：（由佳　(age 20)）正：（由佳　(age 19)）

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD　愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
□1. 書店店頭　□2.他社出版物（　　）
□3.小社出版物　□4. DM　□5.口コミ

パソコンの所有形態（機種名）
□1.個人所有　（　　）
□2.職場にある（　　）
□3.持っていない

本誌を購入した書店　　市（区）　　書店

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事（いくつでも）

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3
パソコン雑誌 ①　②　③
ソフトウェア ①　②　③
パソコン本体 ①　②　③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください（「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください）。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社（学校）名		9	所属（学部）		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ（ハード）技術者　②コンピュータ（ソフト）技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

広告資料請求No. / 希望項目												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払

麹町局承認

3908

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ
READER'S SURVEY CARD 愛読者カード

氏名	性別 □1男 □2女	生年月日 19 年 月 日
住所 〒		電話 ( ) −

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（ ）

ユーザー区分
□1プロ
□2マニア
□3一般ユーザー

創刊記念プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払

麹町局承認

3910

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係 行

＊当社で記入

1	SE	2	受付No.

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓 名	5	生年月日	19 年 月 日
6	自宅住所	〒 電話 ( )			
7	会社住所	〒 電話 ( )			

資料送付希望先の番号に○印をつけてください。

式会社

コンピュータのソフト・ハ
てきたエル・エス・アイ ジ
プログラム開発支援ツール
抜群の効率性とROM化へ
準として高い評価を得てき
て、ソフト面ではISDN
管理システムなど、ハード
ターフェースボードなどを
製品開発の基盤となる人材
るのか、編集部は千駄ヶ谷
総務部長を訪ねてみた。

——その理由は何でしょうか？

「やはり我々の教育方針や技術的なモノの考え方を植えつけられることですね。わが社のカラーといいますか、それを一から教えて、より高いレベルのプロの技術者として育てたい、というのが社長をはじめ我々の一貫した考えでして。ですから教育への投資はまったく惜しみません。確かに今、若い人は常識が欠けていたり、社会を甘く見る面を感じることがありますが、私どもでは企業と社会のかかわり、そうい

「特に必要とはしていません。自分で培ってきたものが、我々とうまく合うかどうかですから。つまりソフトに限らずハード面でデジタル回路が得意ということでもいいわけで、一つのポリシーさえ持っていればと思っています。」

——希望としてはいかがですか？

「希望的には20代まででですね。30代以上ではマネジメントができるなど、管理能力があるということが大きな条件になってきます。



〈プロフィール〉
近藤順司郎
こんどうじゅんしろう

昭和16年生まれ。専修大学経済学部卒。卒業後、金融会社に2年勤務。その後、家業である近藤製作所に従事。昭和58年にかねてから知己である社長の勧めで同社に入社。現在、総務部長として社内の取りまとめ的立場にある。「渋谷で、東京で、そして世の中で光っている会社にしたい。」がモットー。

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵　政　省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係 行

2 受付No.

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名

5 生年月日　19　年　月　日

□10文系学生　□11その他(　　　　)　ユーザー

創刊記念プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）
7	会社住所	〒　電話（　　）

資料送付希望先の番号に○印をつけてください。

綴じ込みの1

エル・エス・アイジャパン株式会社

ちょっと気になるこの会社

1979年に設立以来、マイクロコンピュータのソフト・ハード両面から高い技術を追求してきたエル・エス・アイ　ジャパン。特に、同社が開発したプログラム開発支援ツール「LSI−Cコンパイラー」は、抜群の効率性とROM化への最適化を成し遂げ、業界の標準として高い評価を得てきた。他にも、通信を得意分野として、ソフト面ではISDN関連や文字放送システム、勤務管理システムなど、ハード面では回線エミュレータやインターフェースボードなどを手がけている。こうした幅広い製品開発の基盤となる人材をどのように採用・教育しているのか、編集部は千駄ヶ谷にオフィスを構える本社の近藤総務部長を訪ねてみた。

新卒採用が主体で教育に重点

ー現在、社員数はどれぐらいですか？

「この春に新卒者が３名入りまして、45名程になりました。」

ー新卒と中途採用の構成については？

「うちの場合は新卒採用がほとんどです。」

――コンピュータ業界では企業体力をつけるために中途採用を主としている会社が多い中で、あえて新卒を採る、その理由は何でしょうか？

「やはり我々の教育方針や技術的なモノの考え方を植えつけられることですね。わが社のカラーといいますか、それを一から教えて、より高いレベルのプロの技術者として育てたい、というのが社長をはじめ我々の一貫した考えでして。ですから教育への投資はまったく惜しみません。確かに今、若い人は常識が欠けていたり、社会を甘く見る面を感じることがありますが、私どもでは企業と社会のかかわり、そういうところから教育して若さのいい面を引き出していこうと考えています。」

ー中途採用することもあるのですか？

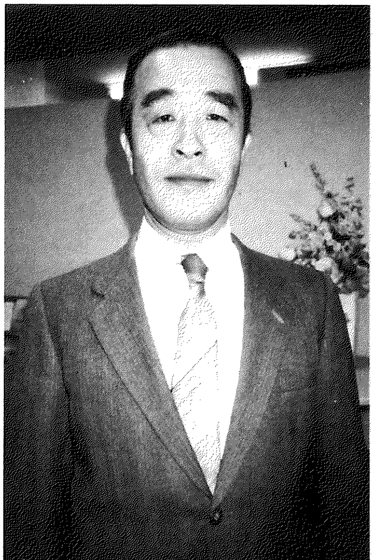
「あります。中途採用の場合は、エキスパート、その道のプロとして採ります。ただ、まったくの未経験者でも採用した実績はあります。新卒でも中途でも、要はやる気で判断しています。」

――例えば中途採用者に最低限必要な知識として、C言語をマスターしているなどの条件はありますか？

「特に必要とはしていません。自分で培ってきたものが、我々とうまく合うかどうかですから。つまりソフトに限らずハード面でデジタル回路が得意ということでもいいわけで、一つのポリシーさえ持っていればと思っています。」

――希望としてはいかがですか？

「希望的には20代までですね。30代以上ではマネジメントができるなど、管理能力があるということが大きな条件になってきます。



〈プロフィール〉
近藤順司郎
こんどうじゅんしろう

昭和16年生まれ。専修大学経済学部卒。卒業後、金融会社に２年勤務。その後、家業である近藤製作所に従事。昭和58年にかねてから知己である社長の勧めで同社に入社。現在、総務部長として社内の取りまとめ的立場にある。「渋谷で、東京で、そして世の中で光っている会社にしたい。」がモットー。

教育の成果は高い定着率が証明

――新卒者を中心に採る貴社は教育にかけてはかなり自信があるようですが、その内容はどうなっていますか？

「新卒の場合は４月に入社してから約２ケ月半から３ケ月かけて教育します。６月の初めには新卒と中途が一同に会して御殿場の国立青年の家で３泊４日の集合研修を行います。この席で社長が会社の方針や考え方を述べたり、新規に人間それぞれが自分の持っている技術を発表し合います。そのほか、社外の講習会でマイクロコンピュータの入門編、ハード・ソフトの基礎、Ｃ言語なども学べます。こうして、６月半ば頃まで研修期間を設けていますが、それ以降も個々のレベルに応じてＯＪＴというかたちで行います。」

――新人の方はそれだけ研修期間あると同期意識も生まれてくるでしょうね。やはりスムーズに仕事に入っていけるようですか？

「それはスムーズにいっています。

また当社の教育の成果は、定着率の高さにも現れています。新卒の場合、家業を継ぐとか結婚するという理由以外では、辞めて他社に移る社員はほとんどいません。わが社では派遣業務も一切していませんし、開発も受託もすべて我々自社内で業務を行っていますから、そういうことも大きいと思いますよ。」

――では中途採用の場合の教育はどうなっているのでしょう。

「技術を持って入社される方は、その道のプロですから、そのまま技術部の即戦力として配属になります。

ただ最近は入社して１、２年後に、もう少し上のレベルの再教育もしていきたいなと考えています。ソフトのプロでもハードウェアの知識を身につけてもらったり、社員一人ひとりが幅をもてるような教育システムを今、模索しているところです。」



社員の努力に報いられる会社に

――近藤総務部長が見られて、社風はどのように感じますか？

「『LSI-Cコンパイラー』に代表されるように、わが社では人に負けない素晴らしい技術をもっていますし、社員がプライドを持って仕事に打ち込める雰囲気があると思います。平均年齢も28歳位と若くて明るい会社です。」

――社員がプライドを持つためには見返りとしての給与も大切ですが……。

「それほど悪くないと思いますよ。業界でもいい方ではないですか。また夏・冬のボーナス以外に、期末賞与が出ます。これはみんなの成果であり、利益が出た分を社員に還元するわけですが、過去ずっと続いています。」

――福利厚生についてはどうですか？

「特に会社で保養施設を持っているということはありませんが、社員一人ひとりに保健や厚生年金をかけている以外に、会社負担でそれに上積みして個人の将来のために年金制度を設けています。また海外研修も毎年１回は必ず数名ずつ行っています。今まで20名以上行っていますが、女性も含まれており、男女の差は待遇面すべてにおいてありません。

――社員間のコミュニケーションはどのように図られているのでしょう。

「社員旅行などは親睦会がありまして、その運営で行っています。昨年は香港、今年はサイパンを３泊４日で予定しています。そのほか、ボウリング大会や野球大会もありますし、新入社員や中途採用者が入った時なども歓迎会を催したり、こうした運営も、すべて社員の自発性によるものなんです。」

――近藤総務部長のお話を伺っていると、社員の意欲には応えようという愛情のようなものがひしひしと感じられますが……。

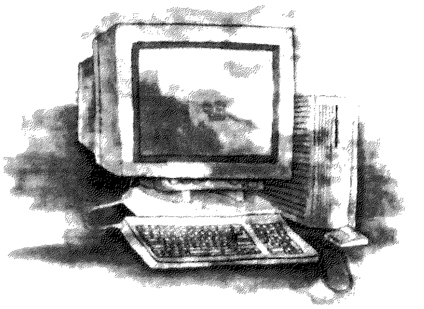
「とにかくうちは、一辺会社に入れたら、自分の子供を育てる事と同じようなものだと考えています。私を含めてうちの幹部は新卒や中途を採った時は喜びを感じて、何とか育てようと考えます。育てること自体が楽しいんですよ。今後はオリジナル製品をさらに磨きをかけて５年先くらいをメドに店頭公開したいと考えています。そのためにも、やる気があって、こういうことしたいという人材にはその場を提供して、うまく能力を伸ばしてあげたいですね。そしてその努力には必ず報いられる会社にしていきたいと考えています。何と言っても、人はかけがえのない財産だと思っていますから。」

●

中途採用者が多く、実力主義の風潮が強いコンピュータ業界で、あえて新卒者にこだわるエル・エス・アイ　ジャパンは稀有な存在に映る。しかし、そのこだわりは定着率の高さに裏打ちされており、家族的な雰囲気を物語っている。これから実力をじっくり磨こうという新卒者、また生涯にわたる就職先を探そうとうい人には格好の場かもしれない。

ELTEX

常にパーフェクトでありたい。



ユーザーや上司が求めているものを機械的に作り上げていく。そんな顧客志向型が、本当のSE・プログラマーと呼べるであろうか。常に思考し続け、より高い価値観を徹底追求し、期待以上のものを創造していく。それが私たちの本当の姿なのだ。提案型ソフトウェアハウスのエルテックスは、そんなSE・プログラマーたちの集まり。たえずパーフェクトなものを求め続けていく集団なのです。

●

人材募集

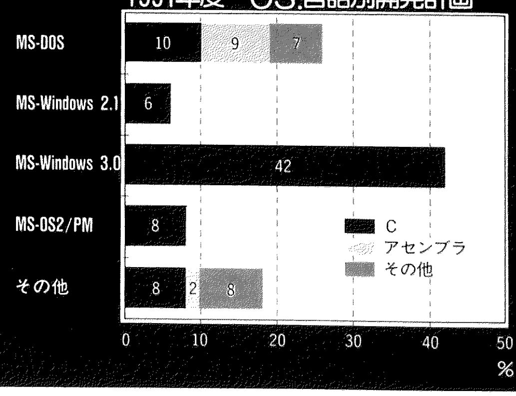
会　社　概　要

設立	昭和60年12月14日
資本金	1,000万円
売上高	2.3億円（平成1年度）
事業内容	パソコンを中心としたシステムの提案・開発・販売 ①CD-ROM・LDなど新しいメディアに対応したシステムの開発 ②ネットワークの端末システムの開発 ③特定業種向けCADの開発・販売 ④基本ソフトの開発
特徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している また、CADを中心とした自社プロダクツにも積極的で、その比率も増加している 基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、OEM製品のBIOS・ユーティリティなどを企画から手掛けている
取引先	三菱電機、三菱自動車、日本航空、旭硝子ほか
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募　集　要　項

職種	①システムエンジニア　②プログラマー
資格	専門卒以上20〜36歳の方 C言語・アセンブラ言語経験者大歓迎 未経験でも好奇心・向学心の旺盛な方は歓迎します
給与	17万円（専門卒初任給）〜46万円（36歳課長、諸手当含む） 24歳例：23万4千円（情報処理1種手当・住宅手当含む） 30歳例：31万6千円（中級SE、情報処理1種・住宅手当含む）
手当	時間外手当 役職手当（5千円〜6万円） 住宅手当（1万5千円） 資格手当（2種：5千円〜特種：2万円） 通勤手当
賞与	賞与年2回（平成1年度実績年間5か月） 決算賞与年1回（平成1年度実績1か月）
勤務地	本社（横浜駅近く） ◇派遣・出向や社外での開発作業はありません
勤務時間	フレックスタイム制（コアタイム10：00〜16：00） 試用期間中は、標準時間9：00〜17：45
休日休暇	完全週休2日制（土日）、祝日、年3回連休（GW、夏期、年末年始）、有給休暇、特別休暇
福利厚生	社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮（制限有）、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書（写真貼付）、職務経歴書を持参し来社してください　郵送も受け付けます
連絡先	〒220　横浜市西区南幸二丁目20番5号　東伸24ビル5階 Tel（045）322-9661　総務課

1991年度　OS.言語別開発計画



8086インストラクション一覧

フラグレジスタ（FLAGSの左から）
- OF：オーバーフローフラグ
- DF：ディレクションフラグ
- IF：割り込み許可フラグ
- TF：トラップフラグ
- SF：サインフラグ
- ZF：ゼロフラグ
- AF：補助キャリーフラグ
- PF：パリティフラグ
- CF：キャリーフラグ

※記号の意味
- -：影響されない
- x：変化する
- ?：不定になる
- 1：セットされる
- 0：クリアされる

オペランド
- reg：汎用レジスタ
- mem：メモリ
- acc：アキュムレータ
- imm：イミディエイトデータ
- sreg：セグメントレジスタ

※記号の意味
- ×：不可
- ○：ワードオペランド使用可能
- ◎：バイトまたはワードオペランド使用可能

データ転送命令

ニーモニック	書式	機能	FLAGS
MOV	MOV 〈dest〉,〈src〉	〈src〉で指定されるオペランドの内容を〈dest〉で指定されるオペランドに転送する	---------
XCHG	XCHG 〈dest〉,〈src〉	〈dest〉と〈src〉の内容を交換する	---------
PUSH	PUSH 〈src〉	〈src〉で指定されるオペランドの内容をスタックに退避する	---------
POP	POP 〈dest〉	スタックから取り出した内容を〈dest〉で指定されるオペランドに転送する	---------
XLAT	XLAT	DS:BXに置かれた変換テーブルに従い，ALレジスタの内容を変換する	---------
IN	IN 〈dest〉,〈port〉	〈port〉で指定されるI/Oポートの内容を〈dest〉で指定されるオペランドに転送する	---------
OUT	OUT 〈port〉,〈src〉	〈port〉で指定されるI/Oポートに〈src〉で指定されるオペランドの内容を転送する	---------
LEA	LEA 〈dest〉,〈src〉	〈src〉で指定されるオペランドのオフセットを〈dest〉で指定されるオペランドに転送する	---------
LDS	LDS 〈dest〉,〈src〉	〈src〉で指定されるダブルワードのポインタのうちセグメントセレクタをDSレジスタに，オフセットを〈dest〉で指定されるオペランドに転送する	---------
LES	LES 〈dest〉,〈src〉	〈src〉で指定されるダブルワードのポインタのうちセグメントセレクタをESレジスタに，オフセットを〈dest〉で指定されるオペランドに転送する	---------
LAHF	LAHF	フラグワードの下位バイトの内容をAHレジスタに転送する	---------
SAHF	SAHF	フラグワードの下位バイトにAHレジスタの内容を転送する	----xxxxx
PUSHF	PUSHF	スタックにフラグワードの内容を退避する	---------
POPF	POPF	スタックから取り出した内容をフラグワードに転送する	xxxxxxxxx

MOV

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	×	◎	○
mem	◎	×	◎$_1$	◎	○
acc	×	◎$_1$	×	×	×
imm	×	×	×	×	×
sreg	○$_2$	○$_2$	×	×	×

XCHG

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	○	×	×
mem	◎	×	×	×	×
acc	○	×	×	×	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

PUSH

〈src〉 reg	mem	acc	imm	sreg
○	○	×	◎$_3$	○

POP

〈dest〉 reg	mem	acc	imm	sreg
○	○	×	×	○$_4$

IN

〈dest〉 reg	mem	acc	imm	sreg
×	×	◎	×	×

OUT

〈src〉 reg	mem	acc	imm	sreg
×	×	◎	×	×

LEA

dest＼src	reg	mem	acc	imm	sreg
reg	×	◎$_5$	×	×	×
mem	×	×	×	×	×
acc	×	×	×	×	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

LDS/LES

dest＼src	reg	mem	acc	imm	sreg
reg	×	○$_6$	×	×	×
mem	×	×	×	×	×
acc	×	×	×	×	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

注1：ダイレクトアドレッシングのみ
注2：CSレジスタを除く
注3：バイトデータはワードデータに符号拡張される
注4：CSレジスタは除く
注5：オペランドのサイズは意味を持たない
注6：ダブルワード

算術演算命令

ニーモニック	書式	機能	FLAGS
ADD	ADD 〈dest〉,〈src〉	〈dest〉と〈src〉によって指定される2つのオペランドの内容を加算し，その結果を〈dest〉で指定されるオペランドに転送する	x---xxxxx
ADC	ADC 〈dest〉,〈src〉	〈dest〉と〈src〉によって指定される2つのオペランドの内容およびCFの内容を加算し，その結果を〈dest〉で指定されるオペランドに転送する	x---xxxxx
INC	INC 〈op〉	オペランドの内容に1を加算する	x---xxxx-
AAA	AAA	1バイトの10進数どうしの加算結果を1バイトの10進数に補正する	?---??x?x
DAA	DAA	1バイトのパックBCDどうしの加算結果をパックBCD形式に補正する	?---xxxxx
SUB	SUB 〈dest〉,〈src〉	〈dest〉で指定されるオペランドの内容から〈src〉で指定されるオペランドの内容を減算し，その結果を〈dest〉で指定されるオペランドに転送する	x---xxxxx
SBB	SBB 〈dest〉,〈src〉	〈dest〉で指定されるオペランドの内容から〈src〉で指定されるオペランドの内容およびCFを減算し，その結果を〈dest〉で指定されるオペランドに転送する	----xxxxx
DEC	DEC 〈op〉	〈op〉で指定される内容より1減算する	x---xxxx-
NEG	NEG 〈op〉	〈op〉で指定される内容の2の補数をとる	x---xxxxx
CMP	CMP 〈dest〉,〈src〉	〈dest〉と〈src〉によって指定される2つのオペランドの内容を比較する	x---xxxxx
AAS	AAS	1バイトの10進どうしの減算結果を1バイトの10進数に補正する	?---??x?x
DAS	DAS	1バイトのパックBCDどうしの減算結果をパックBCD形式に補正する	?---xxxxx
MUL	MUL 〈src〉	バイトまたはワードデータの符号なし乗算を行う	x---????x
IMUL	IMUL 〈src〉	バイトまたはワードデータの符号付き乗算を行う	x---????x
AAM	AAM	2進数から10進数への変換を行う	x---xx?x?
DIV	DIV 〈src〉	“ワード÷バイト”または“ダブルワード÷ワード”の符号なし除算を行う	?---?????
IDIV	IDIV 〈src〉	“ワード÷バイト”または“ダブルワード÷ワード”の符号付き除算を行う	?---?????
AAD	AAD	後に実行される除算命令のために，2バイト2桁の10進数を2進数に変換する	?---xx?x?
CBW	CBW	ALレジスタのバイトデータをワードに符号拡張する	---------
CWD	CWD	AXレジスタのワードデータをダブルワードに符号拡張する	---------

ADD/ADC/SUB/SBB/CMP

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	×	◎$_1$	×
mem	◎	×	×	◎$_1$	×
acc	×	×	×	◎	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

INC/DEC/NEG

〈op〉 reg	mem	acc	imm	sreg
◎	◎	×	×	×

MUL/IMUL/DIV/IDIV

〈src〉 reg	mem	acc	imm	sreg
◎	◎	×	×	×

注1：ワードオペランドに対してバイトのイミディエイトデータを操作することができる．この場合バイトデータは16ビットに符号拡張された上で操作される．

ストリング命令

ニーモニック	書式	機能	FLAGS
MOVS	MOVSB	メモリ間転送を行う（オペランドのバイト指定）	---------
	MOVSW	メモリ間転送を行う（オペランドのワード指定）	
	MOVS 〈dummy1〉,〈dummy2〉	メモリ間転送を行う（〈dummy1〉，〈dummy2〉によるサイズ指定）	
CMPS	CMPSB	メモリ間のデータの比較を行う（オペランドのバイト指定）	x---xxxxx
	CMPSW	メモリ間のデータの比較を行う（オペランドのワード指定）	
	CMPS 〈dummy1〉,〈dummy2〉	メモリ間のデータの比較を行う（〈dummy1〉，〈dummy2〉によるサイズ指定）	
SCAS	SCASB	メモリとアキュムレータの比較を行う（オペランドのバイト指定）	x---xxxxx
	SCASW	メモリとアキュムレータの比較を行う（オペランドのワード指定）	
	SCAS 〈dummy〉	メモリとアキュムレータの比較を行う（〈dummy〉によるサイズ指定）	
LODS	LODSB	メモリ中のデータをアキュムレータに転送する（オペランドのバイト指定）	---------
	LODSW	メモリ中のデータをアキュムレータに転送する（オペランドのワード指定）	
	LODS 〈dummy〉	メモリ中のデータをアキュムレータに転送する（〈dummy〉によるサイズ指定）	
STOB	STOBS	アキュムレータのデータをメモリに転送する（オペランドのバイト指定）	---------
	STOSW	アキュムレータのデータをメモリに転送する（オペランドのワード指定）	
	STOS 〈dummy〉	アキュムレータのデータをメモリに転送する（〈dummy〉によるサイズ指定）	

リピート・プリフィックス（ストリング命令と組み合せて使用）

ニーモニック	機能	FLAGS
REP	MOVS命令, STOS命令, INS命令, OUTS命令の前に付けることにより, これらの命令を繰り返し実行する	---------
REPE/REPZ	CMPS命令, SCAS命令の前に付けることにより, これらの命令を繰り返し実行する	---------
REPNE/REPNZ	CMPS命令, SCAS命令の前に付けることにより, これらの命令を繰り返し実行する	---------

論理演算命令

ニーモニック	書式	機能	FLAGS
NOT	NOT 〈op〉	オペランドのすべてのビットを反転する	---------
AND	AND 〈dest〉,〈src〉	2つのオペランドのビットごとの論理積をとり, 演算結果を〈dest〉で指定されるオペランドに転送する	0---xx?x0
OR	OR 〈dest〉,〈src〉	2つのオペランドのビットごとの論理和をとり, 演算結果を〈dest〉で指定されるオペランドに転送する	-0--xx?x0
XOR	XOR 〈dest〉,〈src〉	2つのオペランドのビットごとの排他的論理和をとり, 演算結果を〈dest〉で指定されるオペランドに転送する	0---xx?x0
TEST	TEST 〈dest〉,〈src〉	2つのオペランドのビットごとの論理積をとる, この演算結果は破棄されフラグのみが変化する	0---xx?x0

AND/OR/XOR/TEST

dest \ src	reg	mem	acc	imm	sreg
reg	◎	◎	×	◎1	×
mem	◎	×	×	◎1	×
acc	×	×	×	◎	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

NOT

〈op〉 reg	mem	acc	imm	sreg
◎	◎	×	×	×

注1：ワードオペランドに対してバイトのイミディエイトデータを操作することができる. この場合バイトデータは16ビットに符号拡張された上で操作される.

シフト・ローテート命令（FLAGSのOFは〈count〉に1を指定した場合のみ変化する）

ニーモニック	書式	機能	FLAGS
SHL/SAL	SHL 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ左シフトする	---------
	SAL 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ左シフトする	
SHR	SHR 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ論理右シフトする	x---xx?xx
SAR	SAR 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ算術右シフトする	0---xx?xx
ROL/ROR	ROL 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ左ローテートする	--------x
	ROR 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ右ローテートする	
RCL/RCR	RCL 〈op〉,〈count〉	〈op〉で指定されるオペランドおよびCFを〈count〉で指定される回数だけ左ローテートする	x-------x
	RCR 〈op〉,〈count〉	〈op〉で指定されるオペランドおよびCFを〈count〉で指定される回数だけ右ローテートする	

SHL / SAL / SHR / SAR / ROL / ROR / RCL / RCR

〈op〉 reg	mem	acc	imm	sreg
◎	◎	×	×	×

分岐命令

ニーモニック	書式	機能	FLAGS
CALL	CALL 〈op〉	〈op〉で指定されるサブルーチンに分岐する	---------
RET	RET	CALL命令によって呼び出されたサブルーチンから制御を戻す	---------
	RET 〈op〉	CALL命令によって呼び出されたサブルーチンから制御を戻し, SPレジスタに〈op〉が加算される	
JMP	JMP 〈op〉	〈op〉で指定されるロケーションに分岐する	---------
Jcondition	JE 〈op〉	ZFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	---------
	JZ 〈op〉	ZFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JL 〈op〉	SFとOFが等しくない場合, 〈op〉で指定されるロケーションに分岐する	
	JNGE 〈op〉	SFとOFが等しくない場合, 〈op〉で指定されるロケーションに分岐する	
	JLE 〈op〉	SFとOFが等しくない, またはZFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNG 〈op〉	SFとOFが等しくない, またはZFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JB 〈op〉	CFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNAE 〈op〉	CFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JBE 〈op〉	CFがセットされている, またはZFがセットされて場合, 〈op〉で指定されるロケーションに分岐する	
	JNA 〈op〉	CFがセットされている, またはZFがセットされて場合, 〈op〉で指定されるロケーションに分岐する	
	JP 〈op〉	PFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
Jcondition	JPE 〈op〉	PFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	---------
	JO 〈op〉	OFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JS 〈op〉	SFがセットされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNE 〈op〉	ZFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNZ 〈op〉	ZFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNL 〈op〉	SFとOFが等しい場合, 〈op〉で指定されるロケーションに分岐する	
	JGE 〈op〉	SFとOFが等しい場合, 〈op〉で指定されるロケーションに分岐する	
	JNLE 〈op〉	SFとOFが等しく, かつZFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JG 〈op〉	SFとOFが等しく, かつZFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNB 〈op〉	CFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JAE 〈op〉	CFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNBE 〈op〉	CFとZFがともにクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JA 〈op〉	CFとZFがともにクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNP 〈op〉	PFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JPO 〈op〉	PFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNO 〈op〉	OFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JNS 〈op〉	SFがクリアされている場合, 〈op〉で指定されるロケーションに分岐する	
	JCXZ 〈op〉	CXレジスタが0の場合, 〈op〉で指定されるロケーションに分岐する	
LOOP	LOOP 〈op〉	CXレジスタから1を引いた結果, 0にならなければ〈op〉で指定されるロケーションに分岐する	---------
LOOPE/LOOPZ	LOOPE 〈op〉 LOOPZ 〈op〉	ZFがセットされていてかつCXレジスタから1を引いた結果, 0にならなければ〈op〉で指定されるロケーションに分岐する	---------
LOOPNE/LOOPNZ	LOOPNE 〈op〉 LOOPNZ 〈op〉	ZFがセットされていてかつCXレジスタから1を引いた結果, 0になれば〈op〉で指定されるロケーションに分岐する	---------

割り込み命令

ニーモニック	書式	機能	FLAGS
INT/INTO	INT 〈op〉	〈op〉で指定されるベクトルの割り込みを発生する	-00------
	INTO	OFがセットされていたら, オーバーフロー割り込みを発生する	
IRET	IRET	割り込み処理ルーチンからもとのプログラムに制御を戻す	-00------

プロセッサ制御命令

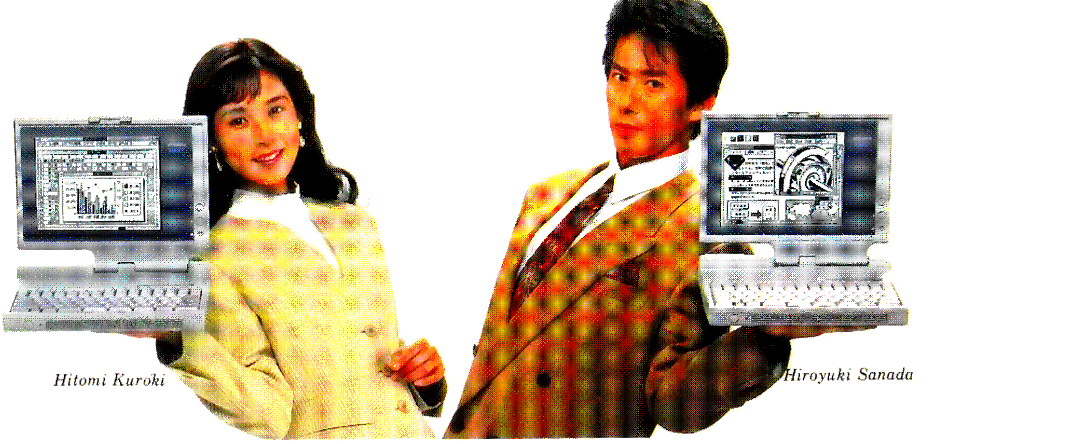
ニーモニック	書式	機能	FLAGS
STC	STC	CFをセットする	--------1
CLC	CLC	CFをクリアする	--------0
CMC	CMC	CFを反転する	--------x
STD	STD	DFをセットする	-1-------
CLD	CLD	DFをクリアする	-0-------
STI	STI	IFをセットする	--1------
CLI	CLI	IFをクリアする	--0------
HLT	HLT	CPUをホールド状態にする	---------
WAIT	WAIT	BUSY端子がHighになるまで命令の実行を中止する	---------
ESC	ESC	拡張プロセッサ命令	---------
LOCK	LOCK	LOCKプリフィックスの付けられた命令の実行中, LOCK端子をアサートする	---------
NOP	NOP	ノーオペレーション	---------

セグメントオーバーライドプリフィックス

ニーモニック	書式	機能	FLAGS
segment:	CS: 〈op〉 DS: 〈op〉 ES: 〈op〉 SS: 〈op〉	メモリオペランドに使用されるセグメントレジスタを〈op〉によって指定されたオペランドに変更する	---------

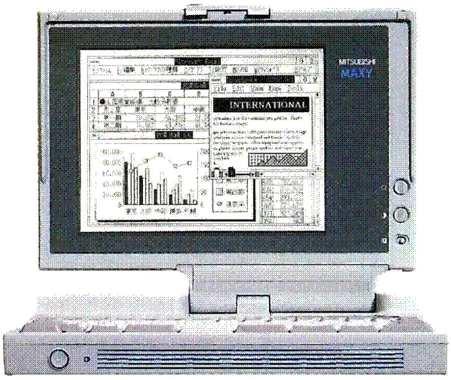
MITSUBISHI

技術がつくる高度なふれあい SOCIO-TECH



片手に乗った、スーパービジネス。

ライバルに差をつける、32ビットノートパソコン。



スーパービジネスは今、MAXYNOTE386から始まる。

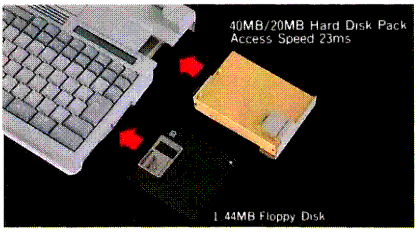
「時代の最先端を切り開くビジネスエリートのために。」スーパービジネスに対応する、MAXYNOTE386の誕生です。A4ファイルサイズのコンパクトボディに、高速32ビットCPU386™SX(16MHz)を搭載。さらにノート初の高速演算プロセッサ387™SXも実装可能(オプション)。広範なビジネスの高速処理を実現します。



写真提供 インテルジャパン(株)

着脱式大容量ハードディスクパックで、思いのままに、スーパービジネス。

片手でポンッと出し入れ自在、フロッピー感覚の超小形2.5インチハードディスクパック(40MB/20MB)を内蔵。大容量情報を効率的に、思い通りにコントロールできます。さらに、統合操作環境MS-Windows、統合ソフトMAXYMATEなどを標準装備。ビジネスパワーを高める即戦力ツールとして即座に活用できます。



MAXYSTATIONⅡで、さらに拡がる、スーパービジネス。

本体一体形のコンパクトな拡張ユニットMAXY STATIONⅡをオプションで用意。PC/ATフルサイズカードが2枚利用可能です。また、100MBハードディスクも内蔵可能。PC/ATの豊富なハードウェア資産を活用した、多彩なネットワーク構築など、デスクトップに迫る豊かな拡張力を提供します。


多彩なシステムアップを構築する拡張ユニットMAXYSTATIONⅡ(拡張カバー装着状態)

●高性能CPU386™SX(16MHz)搭載、387™SX(オプション)●RAM標準1MB(最大5MB)●FDD 3.5インチ×1●HDD 2.5インチ、交換自在なハードディスクパック方式(40MB/20MB)●非同期内蔵モデム実装可(オプション)●RS-232C×1(最大2)●PC/ATフルサイズカード2枚実装可(オプション)●パワーONで瞬時に画面を再現するレジューム機能●MS-DOS、MS-Windows、MAXYMATE、MAXYメニューシステムを標準装備

AXdoNext

世界標準機・三菱AXパソコン

MAXYNOTE386

標準価格●A12(HDD 20MBモデル)478,000円(税別)●A14(HDD 40MBモデル)578,000円(税別)〈この他に省スペースデスクトップタイプもあります〉

ニーズから発想する三菱情報通信システム Think&Link

AX THE ARCHITECTURE EXTENDED-AX

MS-DOS、MS-Windowsは米国マイクロソフト社、386™SX、387™SXは米国インテル社、PC/ATは米国IBM社、MAXY、MAXYSTATION、MAXYMATEは三菱電機(株)の登録商標です。MAXYメニューシステムは、(株)エー・ジー・ソフトのAZ-MOS(エージーモス)をベースとしたメニューシステムです。AZ-MOSは(株)エージーソフトと(株)アイシーエムが開発したインストール方式「IBFファイル(インストールバッチファイル)システム」を採用しています。

世界標準化を推進するAX。日本のソフトウェアメーカ、ハードウェアメーカ、流通各社によるAX協議会★のもと、OS、キーボード、画面、プリンタ部を規格化。国内・海外の豊富なソフトや周辺機器の共通利用を実現します。★1990年12月現在参加553社

三菱電機株式会社

お問合せは…三菱電機株式会社 〒100東京都千代田区丸の内2-2-3(三菱電機ビル) 本社情報処理機器事業部計画部(03)3218-3285 本社情報処理機器部(03)3218-3515 本社システム営業部(03)3218-3544 本社情報通信システム計画部(03)3218-3236 本社電機事業部(03)3218-3158 ●札幌(011)212-3739 ●仙台(022)264-5656 ●大宮(048)653-0261 ●横浜(045)212-3521 ●新潟(025)241-7220 ●金沢(0762)52-8821 ●名古屋(052)565-3246 ●大阪(06)347-2421 ●広島(082)248-5330 ●高松(0878)25-0009 ●福岡(092)721-2346

カタログ請求券 AXパソコン プログラマーズ 9102

あなたの電話番号をご記入ください

T4911777705981

〈資料請求番号P04〉

雑誌コード17777-05